MANUAL DE BASIC

PECOS

SISTEMAS

MANUAL DE BASIC

BASIC-80

MANUAL DE CONSULTA

Manual de consulta del lenguaje BASIC-80 de Microsoft, edición 5.0 y posteriores.

Existen importantes diferencias entre la edición 5.0 del BASIC-80 y las previas (4.51 y anteriores). Si tuviera programas escritos según una edición anterior del BASIC-80, compruebe en el Anexo A los nuevos dispositivos de la 5.0 que pueden afectar la ejecución.

INTRODUCCION

BASIC-80 es la implementación más amplia del BASIC disponible para los microprocesadores 8080 y Z80. En su quinta edición (5.0), el BASIC-80 cumple con las exigencias ANSI para el BASIC, tal como se consignan en el documento BSRX3.60-1978. Toda edición del BASIC-80 consta de tres versiones compatibles hacia arriba: 8K, Ampliada y Disco. Este manual es de consulta para todas las tres versiones del BASIC-80, edición 5.0 y posteriores. Sirve también de consulta para el BASIC-86 y el Compilador BASIC de Microsoft. Actualmente se dispone del BASIC-86 en las versiones Ampliada y Autónoma en Disco, comparables con las versiones del BASIC-80 Ampliada y Autónoma en Disco.

Existen importantes diferencias entre la edición 5.0 del BASIC-80 y las previas (4.51 y anteriores). Si tuviera programas escritos según una edición anterior del BASIC-80, compruebe en el Anexo A los nuevos dispositivos de la 5.0 que pueden afectar la ejecución.

El manual se divide en tres grandes capítulos más un número de apéndices. El Capítulo 1 trata una variedad de temas, en gran medida atinentes a la representación de la información cuando se usa el BASIC-80. El Capítulo 2 contiene la sintaxis y la semántica de cada uno de los comandos y sentencias del BASIC-80, ordenados alfabéticamente. El Capítulo 3 describe todas las funciones intrínsecas del BASIC-80, también ordenadas alfabéticamente.

Los apéndices contienen información sobre los sistemas operativos en particular, más listas de mensajes de error, códigos ASCII y funciones matemáticas; y también información útil sobre subrutinas en lenguaje Assembly y E/S en disco.

CONTENIDO

INTRODUCCION

CAPITULO 1	Información General sobre el BASIC 80
CAPITULO 2	Comandos y Sentencias del BASIC-80
CAPITULO 3	Funciones del BASIC-80
APENDICE A	Nuevos Dispositivos del BASIC-80, Edición 5.0
APENDICE B	Entrada/Salida en Disco BASIC-80
APENDICE C	Subrutinas en Lenguaje Assembly
APENDICE D	BASIC-80 con el Sistema Operativo CP/M
APENDICE E	Conversión de Programas al BASIC-80
APENDICE F	Resumen de Códigos y de Mensajes de Error
APENDICE G	Funciones Matemáticas

INDICE

ABS 3-1
alimentación de línea 1-2, 2-27, 2-31, 2-32, 2,67, 2-68
ALL 2-3, 2-7
Arcotangente 3-3
archivos al azar 2-2, 2-23, 2-29, 2-37, 2-45, 2-52, 3-81, 3-25, A-2, B-7, D-4
archivos protegidos 2 60, A-2, B-2
archivos secuenciales ..... 2 28 a 2-39, 2 32, 2-47, 2 51, 2 68, 3-10, 3-21, B-3
ASC 3-2
ASCII, códigos 3-2, 3-5
ASCII, formato 2-3, 2-28, 2 60
Assembly, subrutinas en lenguaje ..... 2-2, 2-13, 2-48, 3-40, 3-42, C 1
ATN 3-3
AUTO 1-3, 2-1

Boole, operadores de 1-8-2
CALL 2-2, C-5

cassette, cinta 2-5, 2-9
CDBL 3-4
CHAIN 2-3, 2-7
CHR$ 3-5
CINT 3-6
CLEAR 2-4, A-1
CLOAD 2-5
CLOAD* 2-5
CLOAD? 2-5
CLOSE 2-6, B-3, B-8
COMMON 2-3, 2-7
concatenación 1-8-5
constantes 1-5
CONT 2-8, 2-31
control, caracteres 1-4-1
Control-A 2-16
COS 3-7
CP/M 2 35, 2-38, 2-59, 2-60, B-1, D-1
CSAVE 2-9
CSAVE* 2-9
CSNG 3 8
CVD 3-9 B-8
CVI 3-9 B-8
CVS 3-9 B-8

DATA 2 10, 2 57
DEF DN 2-11
DEF USR 2-13, 3-40
DEFDBL 1-6-1, 2-12
DEFINT 1-6 1, 2-12
DEFSNG 1-6-1, 2-12
DEFSTR 1 6-1, 2-12
DEINT C-1
DELETE 1-3 1, 2-3, 2-14
DIM 2-15
división 1-8-1-1

EDIT 1-3, 2-16
edit, modalidad 1-4, 2 16
END 2-6, 2-8, 2 17, 2 24
entero 3-6, 3 12, 3-18
entero, división 1-8 1-1
EOF 3-10, B-3, B-5, D-4
ERASE 2-18
ERL 2-19
ERR 2-19
ERROR 2-20
error, códigos 1-10, 2 19, 2-20, F-1
error, desvío 2-19, 2 20, 2 43, 2-58, B-7
error, mensajes 1-10, F-1
escape 1-14, 2-16
EXP 3-11
exponenciación 1-8, 3-11
expresiones 1-8

FIELD 2-21, B-8
FILES D-3
FIX 3-12,
FOR ... NEXT 2-22, A-1
FRCINT C-1, C-4, D-4
FRE 3 13
funciones 1-8-4, 2 11, 3-1, G-1

GET 2-21, 2-23, B-8, D-4, A-3
GIVABF ..... C-1, a C-2
GOSUB 2-24
GOTO 2-25

HEX$ 3-14
hexadecimal 1-5, 3-14
IF ... GOTO 2-26
IF ... THEN 2 19, 2-26
IF ... THEN ... ELSE 2-26
INP 3-15
INPUT 2-8, 2 21, 2 27, A-1, B-9
INPUT$ 3 16
INPUT≠ 2 28, A-2, B-3
INSTR 3-17
INT 3-12, 3-18
Interrupciones C-7
ISIS-II 2-59

juego de caracteres 1-4

KILL 2-29, B-2

lazos 2-22, 2-65
LEFT$ 3-19
LEN 3-20
LET 2-21, 2-30, B-9
LINE INPUT 2 31
LINE INPUT≠ 2-32, A-2, B-3
línea, Impresora 2-34, 2-36, 2 66, 3 23,A-1
línea, números 1 3, 2-2, 2-73
líneas 1-3
LIST 1-3 1, 2-33

LLIST 2-34
LOAD 2-35, 2-60, B-1
LOC 3-21, B-3, B-5, B-8
LOF D-4
LOG 3-22
lógicos, operadores 1-8-2
LPOS 2-66, 3 23
LPRINT 2-36, 2-66
LPRINT USING 2-36,
LSET 2-37, B-8

MAKINT C-1, C-4, D-4
matrices 1-6-2, 2 5, 2.9, 2-18
matriz, variables 1-6 2, 2-7, 2-15
MBASIC D-1
MERGE 2 3, 2-38, B-2
MID$ 2.39, 3.24
MKD$ 3 25, B-8
MKI$ 3.25, B-8
MKS$ 3.25, B-8
MOD, operador 1.8.1.1
modalidad directa 1.2, 2.26, 2.43
modalidad indirecta 1.2
módulo, aritmética 1.8.1
multiplicación 1.8.1

nivel de comando 1.2
numéricas, constantes 1.5
numéricas, variables 1.6
números al azar 2.53, 3.30

OCT$ 3.26
octal 1.5, 3.26
ON ERROR GOTO 2.43
ON ... GOSUB 2.44
ON ... GOTO 2.44
OPEN 2.6, 2.2, 2.45, B-3, B-8
operadores 1.8
operadores relacionales 1.8.2
OPTION BASE 2.46
OUT 2-47
overflow 1.8.1, 5.11, 3.39, A-1

papel, cinta 2.42
PEEK 2.48, 3.27
POKE 2.48, 3.27
POS 2.66, 3.28
precisión doble 1.5.1, 2.12, 2.49, 3.4, A-1
precisión simple 1.5.1, 2.12, 2.49, 3-5, A-1
PRINT 2.49, A-1
PRINT USING 2.50, A.1
PRINT≠ ..... 2.51, A.2, B. 3
PRINT≠ USING ..... 2.51, A.2, B·5, B.5
PUT 2.51, 2.52, B.8

RANDOMIZE 2.53; 3.30, A.1
READ 2.54, 2.57
REM 2-55

RENUM 2.3, 2.20, 2.56
RESET D-3
resta 1.8.1
RESET D-3
RESTORE 2-57
RESUME 2.58
retorno de carro ..... 1.4, 2.7, 2.31, 2.32, 2.66, a 2.68
RETURN 2.24
RIGHT$ 3.29
RND 2.53, 3.30, A.1
RSET 2.37, B.8
rubout 1.4, 1.9, 2.1
RUN ..... 2.59 a 2.60, B
SAVE 2.35, 2.60, B.1
serie, espacio 2.4, 3.19, A.1,
serie, constantes 1.8.5
serie, funciones ..... 3.9, 3.17 a 3.19, 3.24, 3.29, 3.36, 3.41
serie, operadores 1.8.5
serie, variables ..... 1.6, 2.12, 2.31 a 2.32
SGN 3.31
SIN 3.32
SPACE$ 3.33
SPC 3.34
SQR 3.35
STOP 2.8, 2.17, 2.24, 2.61
STR$ 3.36
STRING$ 3.37
subíndices 1.6, 2.15, 2.46
subrutinas 2.2, 2.24, 2.44, C.1
suma 1.8.1
superposición 2.3
SWAP 2.62
SYSTEM D-4
TAB 3.38
tabulación 1-4
TAN 3.39
TROFF 2.68
TRON 2.63
USR 2.13, 3.40, C.1
USRLOC C-2
VAL 3.41
variables 1-6
VARPTR 3.42
WAIT 2.64
WEND 2.65
WHILE 2.65
WIDTH 2.66, A.1
WIDTH LPRINT 2.66, A.1
WRITE 2.67
WRITE≠ 2.67, A.2, B.3

CAPITULO 1

INFORMACION GENERAL SOBRE EL BASIC-80

1.1 INICIALIZACION

El procedimiento de inicialización variará con las diferentes implementaciones del BASIC-80. Determine, consultando el apéndice correspondiente, cómo se inicializa el BASIC-80 con su sistema operativo.

1.2 MODALIDADES DE OPERACION

Una vez inicializado el BASIC-80 se advierte la indicación "Ok". "Ok" significa que el BASIC-80 está a nivel de comando, esto es, en condiciones de aceptar comandos. En este punto puede usarse el BASIC-80 en una de dos modalidades: directa o indirecta.

En la modalidad directa, los comandos y sentencias BASIC no son precedidas por números de línea. Se ejecutan tal como se ingresan. Los resultados de las operaciones aritméticas y lógicas pueden representarse de inmediato y almacenarse para su posterior utilización, pero las instrucciones, en sí mismas, se pierden tras la ejecución.

Esta modalidad es útil en corrección de errores y para usar el BASIC como una "calculadora" en cálculos rápidos que no requieren un programa completo.

La modalidad indirecta es la usada para ingresar programas. Las líneas del programa se preceden con números de línea y se almacenan en la memoria. El programa almacenado en la memoria se ejecuta ingresando el comando RUN.

1.3 FORMATO DE LA LINEA

Las líneas de un programa BASIC tienen el siguiente formato (los corchetes indican opcional):

nnnnn sentencia BASIC [:sentencia BASIC . . .] ⟨retorno de carro⟩

A opción del programador, en una línea puede colocarse más de una sentencia BASIC, pero cada una de ellas ha de separarse de la anterior con dos puntos (:).

Una línea de un programa BASIC comienza siempre con un número de línea, termina con un retorno de carro y puede contener un máximo de:

72 caracteres en BASIC-80 8K

255 caracteres en BASIC-80 Ampliado y Disco

En las versiones Ampliado y Disco es posible extender una línea lógica en más de una línea física por el uso de la tecla ⟨alimentación de línea⟩ de la terminal. ⟨Alimentación de línea⟩ le permite seguir digitando una línea lógica en la siguiente línea física sin ingresar un ⟨retorno de carro⟩. (En la versión 8K, ⟨alimentación de línea⟩ carece de efecto.)

1.3.1 Números de Línea

Toda línea de un programa BASIC comienza con un número de línea. Los números de línea indican el orden en que las líneas del programa se almacenan en la memoria; se usan también como referencias al bifurcar y editar.

Los números de línea deben estar entre los límites 0 y 65529. En las versiones Ampliado y Disco, puede usarse un punto (.) en los comandos EDIT, LIST, AUTO y DELETE para referirse a la línea corriente.

1.4 JUEGO DE CARACTERES

El juego de caracteres BASIC-80 consta de caracteres alfabéticos, numéricos y especiales.
Los caracteres alfabéticos del BASIC-80 son las letras mayúsculas y minúsculas del alfabeto.
Los caracteres numéricos del BASIC-80 son los dígitos 0 a 9.

El BASIC-80 reconoce los siguientes caracteres especiales y teclas de la terminal:

Carácter	Nombre
	Blanco
;	Punto y coma
=	Signo igual o símbolo de asignación
+	Signo más
−	Signo menos
*	Asterisco o símbolo de multiplicación
/	Barra o símbolo de división
^	Flecha hacia arriba o símbolo de exponenciación
(	Paréntesis a la izquierda
)	Paréntesis a la derecha
%	Porciento
#	Signo numeral (o libra)
$	Signo pesos
!	Signo de exclamación
[	Corchete a la izquierda
]	Corchete a la derecha
,	Coma
.	Punto o punto decimal
'	Apóstrofo
:	Dos puntos
&	Ampersand
?	Signo de interrogación
<	Menor que
>	Mayor que
\	Barra inversa o símbolo de división entera
@	Signo 'at'
_	Subrayado
⟨borre⟩	Elimina el último caracter digitado
⟨escape⟩	Sale de subcomandos en Modalidad Edit. Ver sección 2.16
⟨tab⟩	Transfiere la posición de impresión al siguiente retén de tabulación. Los retenes de tabulación están cada ocho columnas
⟨aliment. de línea⟩	Traslada a la siguiente línea física
⟨retorno de carro⟩	Termina la entrada de una línea

1.4.1 Caracteres de Control

El BASIC-80 cuenta con los siguientes caracteres de control:

Control-A	Ingresa en Modalidad Edit en la línea que se está digitando.
Control-C	Interrumpe la ejecución del programa y vuelve a nivel de comando BASIC-80.
Control-G	Hace sonar el timbre de la terminal.
Control-H	Retroceso. Elimina el último caracter digitado.
Control-I	Tabulación. Hay retenes de tabulación cada ocho columnas.

Control-O	Detiene la salida del programa mientras continúa la ejecución. Un segundo Control-O reanuda la salida.
Control-R	Vuelve a digitar la línea que se está digitando.
Control-S	Suspende la ejecución del programa.
Control-Q	Reanuda la ejecución del programa luego de un Control-S.
Control-U	Elimina la línea que se está digitando.

1.5 CONSTANTES

Constantes son los valores reales que el BASIC usa durante la ejecución. Existen dos tipos de constantes: de serie y numéricas.

Constante de serie es una secuencia de hasta 255 caracteres alfanuméricos entre comillas. Ejemplos de constantes de serie:

"HOLA"
"$25,000.00"
"Cantidad de Empleados"

Las constantes numéricas son números positivos o negativos. Las constantes numéricas no pueden contener comas en el BASIC. Cinco son los tipos de constantes numéricas:

1. Constantes enteras — Números enteros entre —32768 y +32767. Las constantes enteras no tienen puntos decimales.

2. Constantes de punto fijo — Números reales positivos o negativos; esto es, números que contienen puntos decimales.

3. Constantes de punto flotante — Números positivos o negativos representados en formato exponencial (similar a la notación científica). Una constante de punto flotante consiste en un número entero opcionalmente con signo o de punto fijo (la mantisa) seguido por la letra E y un entero opcionalmente con signo (el exponente). El límite permitido a las constantes de punto flotante es

   10^{-38} a 10^{+38}. Ejemplos:

   235.988E—7 = .0000235988

   235E6 = 2359000000

   Las constantes de punto flotante de precisión doble utilizan la letra D en lugar de la E. Ver Sección 1.5.1.

4. Constantes hexadecimales — Números hexadecimales con el prefijo &H. Ejemplos:

   &H76

   &H32F

5. Constantes octales — Números octales con el prefijo &O o &. Ejemplos:

   &O347

   &1234

1.5.1 Formato de Precisión Simple y Doble para Constantes Numéricas

En la versión de 8K del BASIC-80, todas las constantes numéricas son números de precisión simple. Se almacenan con 7 dígitos de precisión y se imprimen con hasta 6 dígitos.

Sin embargo, en las versiones Ampliado y Disco, las constantes numéricas pueden ser números de precisión simple o doble. Con precisión doble, los números se almacenan con 16 dígitos de precisión y se imprimen con hasta 16 dígitos.

Una constante de precisión simple es cualquier constante numérica que tenga:

1. siete o menos dígitos, o
2. formato exponencial con el uso de E, o
3. un signo exclamativo a la derecha (!)

Una constante de precisión doble es cualquier constante numérica que tenga:

1. ocho o más dígitos, o
2. formato exponencial con el uso de D, o
3. un signo numeral (#) a la derecha

Ejemplos:

Constantes de Precisión Simple	**Constantes de Precisión Doble**
46.8	345692811
—7.09E—06	—1.09432D—06
3489.0	3489.0#
22.51	7654321.1234

1.6 VARIABLES

Variables son nombres que se emplean para representar valores que se usan en un programa BASIC. El programador puede asignar explícitamente el valor de una variable, o bien éste puede asignarse como resultado de cálculos del programa. Antes de asignarse un valor a una variable, se presupone que el valor de ésta es cero.

1.6.1 Nombres de Variables y Caracteres de Declaración

Los nombres de las variables BASIC-80 pueden ser de cualquier longitud; no obstante, en la versión 8K, sólo los primeros dos caracteres son significativos. En las versiones Ampliado y Disco, hasta 40 caracteres son significativos. Los caracteres que se permiten en el nombre de una variable son letras y números, permitiéndose el punto decimal en nombres de variables Ampliado y Disco. El primer carácter debe ser una letra. También se permiten caracteres de declaración de tipo especial (ver más abajo).

El nombre de una variable no puede ser una palabra reservada. Las versiones Ampliado y Disco permiten incluir palabras reservadas; la versión 8K, no. Si una variable comienza con FN, se presume que es un llamado a una función definida por el usuario. Las palabras reservadas incluyen todos los comandos, sentencias, nombres de funciones y operadores BASIC-80.

Las variables pueden representar un valor numérico o una serie. Los nombres de las variables de serie se escriben con un signo pesos ($) como último carácter. Por ejemplo: A$ = "INFORME DE VENTAS". El signo pesos es un carácter de declaración de tipo de variable, esto es, "declara" que la variable representará una serie.

En las versiones Ampliado y Disco, los nombres de las variables numéricas pueden declarar valores enteros, de precisión simple o doble. (Todos los valores numéricos en 8K son de precisión simple). Los caracteres de declaración de tipo de estos nombres de variable son los siguientes:

%	Variable entera
!	Variable de precisión simple
#	Variable de precisión doble

El tipo presupuesto (por defecto) de un nombre de variable numérica es precisión simple.
Siguen ejemplos de nombres de variables BASIC-80.

En las versiones Ampliado y Disco:

PI#	declara un valor de precisión doble
MINIMUM!	declara un valor de precisión simple
LIMIT%	declara un valor entero

En las versiones 8K, Ampliado y Disco:

N$	declara un valor de serie
ABC	representa un valor de precisión simple

En las versiones Ampliado y Disco del BASIC-80 existe un segundo método por el cual pueden declararse tipos de variable. Las sentencias BASIC-80 DEFINT, DEFSTR, DEFSNG y DEFDBL pueden incluirse en un programa para declarar los tipos de ciertos nombres de variables. Estas sentencias se describen en detalle en la Sección 2.12.

1.6.2 **Variables de Matriz**

Matriz es un grupo o tabla de valores referenciados por el mismo nombre de variable. Todo elemento de una matriz es referenciado por una variable de matriz con un subíndice entero o expresiones enteras. Un nombre de variable de matriz tiene tantos subíndices como dimensiones haya en la matriz. Por ejemplo V(10) referenciaría un valor de una matriz unidimensional; T(1,4) referenciaría un valor de una matriz bidimensional, etc. El número máximo de dimensiones de una matriz es 255. El número máximo de elementos por dimensión es 32767.

1.7 **CONVERSION DE TIPO**

Cuando fuera necesario, el BASIC convertirá una constante numérica de un tipo a otro. Se tendrán presentes las siguientes reglas y ejemplos:

1. Si se igualara una constante numérica de un tipo a una variable numérica de un tipo diferente, el número se almacenará como el tipo declarado en el nombre de la variable. (Si se igualara una variable de serie a un valor numérico o viceversa se producirá un error de "Falta de concordancia en tipo").

   Ejemplo:

   ```
   10 A% = 23.42
   20 PRINT A%
   RUN
    23
   ```

2. Durante la evaluación de una expresión, todos los operandos de una operación aritmética o relacional se convierten al mismo grado de precisión, esto es, al del operando más preciso. Asimismo, el resultado de una operación aritmética se devuelve a este grado de precisión.

   Ejemplo:

   ```
   10 D# = 6#/7
   20 PRINT D#
   RUN
    .8571428571428571
   ```

   Los cálculos aritméticos se hicieron con precisión doble y el resultado se devolvió en D# como un valor de precisión doble.

   ```
   10 D = 6#/7
   20 PRINT D
   RUN
    .857143
   ```

   Los cálculos aritméticos se hicieron con precisión doble y el resultado se devolvió en D (variable de precisión simple), redondeado e impreso como un valor de precisión simple.

3. Los operadores lógicos (ver Sección 1.8.3) convierten sus operandos a enteros y devuelven un resultado entero. Los operandos deben estar dentro de los límites —32768 a 32767 ó se producirá un error de "overflow" (capacidad excedida).

4. Cuando un valor de punto flotante se convierte a un entero se redondea la porción fraccionaria.

   Ejemplo:

   ```
   10 C% = 55.88
   20 PRINT C%
   RUN
    56
   ```

5. Si se asignara un valor de precisión simple a una variable de precisión doble, sólo serán válidos los primeros siete dígitos, redondeados, del número convertido. Esto se debe al hecho de haberse suministrado sólo siete dígitos de precisión con el valor de precisión simple. El valor absoluto de la diferencia entre el número de precisión doble impreso y el valor original de precisión simple será menor que 6.3E—8 por el valor original de precisión simple.

   Ejemplo:

   ```
   10 A = 2.04
   20 B# = A
   30 PRINT A;B#
   RUN
    2.04  2.039999961853027
   ```

1.8 EXPRESIONES Y OPERADORES

Una expresión puede ser simplemente una constante de serie o numérica, o una variable, o combinar constantes y variables con operadores para producir un valor.

Los operadores cumplen operaciones matemáticas o lógicas en los valores. Los operadores que provee el BASIC-80 pueden dividirse en cuatro categorías:

1. Aritméticos
2. Relacionales
3. Lógicos
4. Funcionales

1.8.1 Operadores Aritméticos

Los operadores aritméticos, en orden de precedencia, son:

Operador	Operación	Expresión de ejemplo
∧	Exponenciación	X∧Y
—	Negación	—X
*,/	Multiplicación	X*Y
	División Punto Flotante	X/Y
+,—	Suma, Resta	X+Y

Utilice paréntesis para modificar el orden en el que se realizan las operaciones. Las operaciones dentro de los paréntesis se realizan en primer término. Dentro de los paréntesis se mantiene el orden usual de las operaciones. Siguen algunos ejemplos de expresiones algebraicas y sus contrapartidas BASIC.

Expresión algebraica	Expresión BASIC
X+2Y	X+Y*2
$X-\frac{Y}{Z}$	X—Y/Z
$\frac{XY}{Z}$	X*Y/Z
$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Y$	(X∧2)∧Y
X^{YZ}	X∧(Y∧Z)
X(—Y)	X*(—Y)

Dos operadores consecutivos deben separarse con paréntesis

1.8.1.1 **División Entera y Aritmética de Módulo.** En las versiones Ampliado y Disco del BASIC-80 se dispone de dos operadores adicionales: división entera y aritmética de módulo.

La división entera se denota con la barra inversa (\). Los operandos se redondean a entero (han de estar en los límites —32768 a 32767) antes de realizarse la división y el cociente se trunca a enteros. Por ejemplo:

```
10\4 = 2
25.68\6.99 = 3
```

La precedencia de la división entera sigue a la multiplicación y división de punto flotante.

La aritmética de módulo se denota con el operador MOD. Da el valor entero que es el resto de una división entera. Por ejemplo:

```
10.4 MOD 4 = 2 (10/4 = 2 con un resto de 2)
25.68 MOD 6.99 = 5 (26/7 = 3 con un resto de 5)
```

La precedencia de la aritmética de módulo sigue a la división entera.

1.8.1.2 **Overflow y División por Cero.** Si durante la evaluación de una expresión se detectara una división por cero, se representa el mensaje de error "División por cero"; como resultado de la división se provee infinitud de máquina con el signo del numerador y continúa la ejecución. Si la evaluación de una exponenciación resultara en cero elevado a una potencia negativa, se representa el mensaje de error "División por cero"; como resultado de la exponenciación se provee infinitud positiva de máquina y la ejecución continúa.

De excederse la capacidad, se representa el mensaje de error "Overflow"; como resultado se provee infinitud de máquina con el signo algebraicamente correcto y la ejecución continúa.

1.8.2 **Operadores Relacionales**

Los operadores relacionales se usan para comparar dos valores. El resultado de la comparación es "verdadero" (—1) o "falso" (0). Luego puede usarse este resultado para tomar una decisión concerniente al flujo del programa (ver IF, Sección 2.26).

Operador	Relación probada	Expresión
=	Igualdad	X=Y
<>	Desigualdad	X<>Y
<	Menor que	X<Y
>	Mayor que	X>Y
<=	Menor que o igual a	X<=Y
>=	Mayor que o igual a	X>=Y

(El signo igual también se usa para asignar un valor a una variable. Ver LET, Sección 2.30.)

Cuando en una expresión se combinan operadores aritméticos y relacionales, siempre se cumplen primeramente los aritméticos. Por ejemplo, la expresión

```
X+Y<(T-1)/Z
```

es verdadera si el valor de X más Y es menor que el valor de T—1 dividido por Z. Más ejemplos:

```
IF SIN(X)<0 GOTO 1000
IF I MOD J<>0 THEN K=K+1
```

Los operadores lógicos realizan pruebas en relaciones múltiples, manipulación de bits u operaciones booleanas.

El operador lógico devuelve un resultado en forma de bits que es "verdadero" (no cero) o "falso" (cero). En una expresión, las operaciones lógicas se realizan después de las aritméticas y relacionales. El resultado de una operación lógica se determina tal como se ilustra en la siguiente tabla. Los operadores se listan en orden de precedencia.

NOT

X	NOT X
1	0
0	1

AND

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

OR

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

IMP

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

EQV

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

Así como los operadores relacionales pueden usarse para tomar decisiones sobre el flujo del programa, los operadores lógicos pueden conectar dos o más relaciones y devolver un valor verdadero o falso a emplear en una decisión. (Ver IF, Sección 2.26).

Por ejemplo:

```
IF D<200 AND F<4 THEN 80
IF I>10 OR K<Q THEN 50
IF NOT P THEN 100
```

Los operadores lógicos funcionan convirtiendo sus operandos a enteros con complemento a dos de dieciséis bits, con signo, en los límites —32768 a +32767. (Si los operandos no estuvieran dentro de estos límites, resulta un error). Si se proveyeran ambos operandos como 0 ó —1, los operadores lógicos revuelven 0 ó —1. La operación dada se realiza en estos enteros en modalidad bit, esto es, cada uno de los bits del resultado es determinado por los correspondientes bits de los dos operandos.

Así es posible usar operadores lógicos para probar bytes con respecto a un modelo particular de bits. Por ejemplo, puede utilizarse el operador AND para "enmascarar" todos salvo uno de los bits de un byte de estado en una vía de entrada/salida de máquina. Puede usarse el operador OR para "intercalar" dos bytes para crear un valor binario en particular. Los siguientes ejemplos ayudarán a demostrar el funcionamiento de los operadores lógicos:

63 AND 16=16	63 = binario 111111 y 16 = binario 10000, así 63 AND 16 = 16
15 AND 14=14	15 = binario 1111 y 14 = binario 1110, así 15 AND 14 = 14 (binario 1110).
—1 AND 8=8	—1 = binario 1111111111111111 y 8 = binario 1000, así —1 AND 8 = 8
4 OR 2=6	4 = binario 1000 y 2 = binario 10, así 4 OR 2 = 6 (binario 110)
10 OR 10=10	10 = binario 1010, así 1010 OR 1010 = 1010 (10)
—1 OR —2=—1	—1 = binario 111111111111111, y —2 = binario 111111111111110, así —1 OR —2 = —1. El complemento de bits de dieciséis ceros es dieciséis unos, que es la representación complemento a dos de —1.
NOT X=—(X+1)	El complemento a dos de cualquier entero es el complemento de bit más uno.

1.8.4 **Operadores Funcionales**

Se usa una función en una expresión para llamar a una operación predeterminada que ha de realizarse en un operando. El BASIC-80 tiene funciones "intrínsecas" que residen en el sistema, tales como SQR (raíz cuadrada) o SIN (seno). Todas las funciones intrínsecas del BASIC-80 se describen en el Capítulo 3.

El BASIC-80 permite asimismo funciones "definidas por el usuario", que escribe el programador. (Ver DEF FN, Sección 2.11).

1.8.5 **Operaciones de Series**

Las series pueden concatenarse usando +. Por ejemplo:

```
10 A$="NOMBRE" : B$="ARCHIVO"
20 PRINT A$ + B$
30 PRINT "NUEVO" + A$ + B$
RUN
NOMBREARCHIVO
NUEVO NOMBREARCHIVO
```

Las series pueden compararse usando los mismos operadores relacionales que se usan con los números:

= <> < > <= >=

Las series se comparan tomando un carácter a la vez de cada una de las series y comparando los códigos ASCII. Si todos éstos fueran los mismos las series son iguales. Si difirieran, el número de código más bajo precede al más alto. Si, durante la comparación de series, se alcanzara el fin de una de ellas, se dice que la más corta es la menor. Los blancos a la derecha y a la izquierda son significativos.

Ejemplos:

"AA"<"AB"
"NOMBREARCHIVO" = "NOMBREARCHIVO"
"X&">"X≠"
"CL">"CL"
"kg">"KG"
"SMYTH"<"SMYTHE"
B$<"9/12/78" en donde B$ = "8/12/78"

Así, las comparaciones de series pueden usarse para probar valores de series o con series alfabéticas. Todas las constantes de serie usadas en expresiones de comparación deben encerrarse entre comillas.

1.9 EDICION DE LA ENTRADA

Si se ingresara un caracter incorrecto mientras se digita una línea, puede eliminarse con la tecla RUBOUT (borre) o con Control-H. RUBOUT circunda el o los caracteres eliminados con barras inversas y Control-H tiene el efecto de retroceder sobre un caracter y borrarlo. Una vez eliminados uno o más caracteres, siga digitando la línea tal como lo desee.

Para eliminar una línea que está en el proceso de ser digitada, digite Control-U. Después de eliminarse la línea se ejecuta automáticamente un retorno de carro.

Para corregir líneas de un programa que está en la memoria actualmente, vuelva a digitar la línea usando el mismo número de línea. El BASIC-80 reemplazará automáticamente la línea anterior con la nueva.

En las versiones Ampliado y Disco del BASIC-80 se proveen posibilidades más refinadas de edición. (Ver EDIT, Sección 2.16).

Para eliminar todo el programa que reside actualmente en la memoria, ingrese el comando NEW. (Ver Sección 2.41). Usualmente se usa NEW para borrar la memoria antes de ingresar un nuevo programa.

1.10 MENSAJES DE ERROR

Si el BASIC-80 detecta un error que termina la ejecución del programa se imprime un mensaje de error. En la versión de 8K, sólo se imprime el código de error. En las versiones Ampliado y Disco, se imprime el mensaje completo. En el Apéndice J podrá observar una lista completa de los códigos y mensajes de error del BASIC-80.

CAPITULO 2

COMANDOS Y SENTENCIAS DEL BASIC-80

En este capítulo se describen todos los comandos y sentencias del BASIC-80. Cada una de las descripciones adopta el siguiente formato:

Formato: Da el formato correcto de la instrucción. Ver a continuación la notación del formato.
Versiones: Lista las versiones del BASIC-80 en las cuales se dispone de la instrucción.
Propósito: Informa el uso de la instrucción.
Observaciones: Describe en detalle cómo se usa la instrucción.
Ejemplo: Ilustra programas o segmentos de programas que demuestran el uso de la instrucción.

Notación del Formato:

Toda vez que se dé el formato de una sentencia o comando son de aplicación las siguientes reglas:

1. Los ítems en letras mayúsculas se ingresarán tal como se ilustran.
2. El usuario debe proveer los ítems en letras mayúsculas encerrados entre corchetes angulares (⟨ ⟩).
3. Los ítems entre corchetes ([ ]) son opcionales.
4. Todos los signos de puntuación, salvo corchetes angulares y corchetes (comas, paréntesis, punto y coma, guiones, signos igual) deben incluirse en donde se ilustran.
5. Los ítems seguidos por puntos suspensivos (. . .) pueden repetirse cualquier número de veces (hasta la longitud de la línea).
6. Los ítems separados por una barra vertical (|) se excluyen entre sí; elija uno.

2.1 AUTO

Formato: Auto [⟨número de línea⟩[,⟨incremento⟩]]
Versiones: Ampliado, Disco.
Propósito: Generar automáticamente un número de línea después de todo retorno de carro.
Observaciones: AUTO comienza a numerar en ⟨número de línea⟩ e incrementa todo subsiguiente número de línea en ⟨incremento⟩. El valor por defecto de ambos valores es 10. Si ⟨número de línea⟩ fuera seguido por una coma pero no se especificara ⟨incremento⟩, se presupone el último incremento especificado en un comando AUTO.

Si AUTO generara un número de línea que ya se está usando, se imprime un asterisco después del número para advertir al usuario que cualquier entrada reemplazará a la línea existente. Sin embargo, la digitación de un retorno de carro inmediatamente después del asterisco resguardará la línea y generará el siguiente número de línea.

AUTO termina digitando Control-C. No se resguarda la línea en la que se digita Control-C. Luego de digitado Control-C, el BASIC vuelve a nivel de comando.

Ejemplo:
AUTO 100,50 Genera los números de línea 100, 150, 200 . . .
AUTO Genera los números de línea 10, 20, 30, 40 . . .

2.2 CALL

Formato: CALL ⟨nombre variable⟩ [(⟨lista argumentos⟩)]
Versiones: Ampliado, Disco.
Propósito: Llamar una subrutina en lenguaje assembly.

Observaciones: La sentencia CALL es una manera de transferir el flujo de un programa a una subrutina en lenguaje assembly. (Ver asimismo, función USR, Sección 3.40.)

⟨nombre variable⟩ contiene una dirección que es el punto inicial en memoria de la subrutina. ⟨nombre variable⟩ no puede ser un nombre de variable de matriz. ⟨lista argumentos⟩ contiene los argumentos que se transfieren a la subrutina en lenguaje assembly.

La sentencia CALL genera la misma secuencia de llamado usado por los compiladores FORTRAN, COBOL y BASIC de Microsoft.

Ejemplo:

```
110 MY ROUT=&HDOOO
120 CALL MY ROUT(I,J,K)
.
.
.
```

2.3 **CHAIN**

Formato: CHAIN [MERGE] ⟨nombrearchivo⟩ [,⟨número línea exp⟩]
[,ALL] [,DELETE⟨límite⟩]]

Versión: Disco.

Propósito: Llamar a un programa y transferirles variables desde el programa corriente.

Observaciones: ⟨nombrearchivo⟩ es el nombre del programa que se llama. Ejemplo:

```
CHAIN"PROG1"
```

⟨número línea exp⟩ es un número de línea o una expresión que evalúa un número de línea del programa llamado. Es el punto inicial de ejecución del programa llamado. Si se omitiera, la ejecución comienza en la primer línea. Ejemplo:

```
CHAIN"PROG1",1000
```

Un comando RENUM no afecta ⟨número línea exp⟩.

Con la opción ALL, toda variable del programa corriente se transfiere al programa llamado. Si se omitiera la opción ALL, el programa corriente debe contener una sentencia COMMON para listar las variables que se transfieren. Ver Sección 2.7. Ejemplo:

```
CHAIN"PROG1",1000,ALL
```

Si se incluyera la opción MERGE, esto permite traer una subrutina al programa BASIC como una superposición. Esto es, se lleva a cabo una operación MERGE con el programa corriente y el llamado. El programa llamado debe ser un archivo ASCII si ha de ser intercalado (MERGE). Ejemplo:

```
CHAIN MERGE"OVRLAY",1000
```

Luego de traída una superposición, usualmente es deseable eliminarla para que pueda traerse una nueva superposición. Para hacer esto, use la opción DELETE. Ejemplo:

```
CHAIN MERGE"OVRLAY2",1000,DELETE 1000-5000
```

El comando RENUM afecta los números de línea de ⟨límite⟩.

NOTA: El Compilador BASIC Microsoft no soporta las opciones ALL, MERGE y DELETE para CHAIN. Si deseara mantener compatibilidad con el compilador BASIC se recomienda utilizar COMMON para transferir variables y no usar superposiciones.

NOTA: Si se omitiera la opción MERGE, CHAIN no preserva tipos de variables o funciones definidas por el usuario para su uso por el programa encadenado. Esto es, en el programa encadenado ha de expresarse nuevamente toda sentencia DEFINT, DEFSGn, DEFDBL, DEFSTR o DEFFN que contenga variables compartidas.

2.4 **CLEAR**

Formato: CLEAR [,[⟨expresión1⟩] [,⟨expresión2⟩]]

Versiones: 8K, Ampliado, Disco.

Propósito: Posicionar todas las variables numéricas en cero y todas las variables de serie en nulos y, opcionalmente, posicionar el fin de la memoria y el total de espacio de apilamiento.

Observaciones: ⟨expresión1⟩ es una ubicación de memoria que, de especificarse, posiciona la ubicación más alta disponible para su uso por el BASIC-80.
⟨expresión2⟩ separa espacio de apilamiento para el BASIC. El valor por defecto es 256 bytes o un octavo de la memoria disponible, el que fuera menor.

NOTA: En versiones anteriores del BASIC-80, ⟨expresión1⟩ posiciona el total de espacio de serie, y ⟨expresión2⟩, el fin de la memoria. BASIC-80, edición 5.0 y posteriores, asigna dinámicamente espacio de serie. Se produce un error de "Falta espacio de serie" solamente si no hubiera memoria libre para su uso por el BASIC.

Ejemplos:
```
CLEAR
CLEAR ,32768
CLEAR,,2000
CLEAR,32768,2000
```

2.5 CLOAD

Formatos:
```
CLOAD ⟨nombrearchivo⟩
CLOAD? ⟨nombrearchivo⟩
CLOAD* ⟨nombre matriz⟩
```

Versiones: 8K (cassette), Ampliado (cassette).

Propósito: Cargar un programa o una matriz desde cinta de cassette en memoria.

Observaciones: CLOAD ejecuta un comando NEW antes de cargar el programa desde cinta de cassette. ⟨nombrearchivo⟩ es la expresión de serie o el primer caracter de la expresión de serie especificado al ser el programa objeto de CSAVE.

CLOAD? verifica cintas comparando el programa corrientemente en memoria con el archivo en cinta que tiene el mismo nombrearchivo. Si fueran los mismos, BASIC-80 imprime Ok. Si no lo fueran, BASIC-80 imprime NO GOOD.

CLOAD* carga una matriz numérica que ha sido resguardada en cinta. Los datos en cinta se cargan en la matriz llamada ⟨nombre matriz⟩ especificada al ser la matriz objeto de CSAVE*.

CLOAD y CLOAD? se ingresan siempre a nivel de comando como comandos en modalidad directa.

CLOAD* puede ingresarse a nivel de comando o usarse como una sentencia de programa. Corrobore que la matriz haya sido DIMensionada antes de ser cargada. BASIC-80 siempre vuelve a nivel de comando después de ejecutarse CLOAD, CLOAD? o CLOAD*. Antes de ejecutar CLOAD, compruebe que la grabadora de cassette esté adecuadamente conectada y en modalidad 'Play' y que la cinta esté correctamente posicionada.

Ver asimismo CSAVE, Sección 2.9.

NOTA: CLOAD y CSAVE no se incluyen en todas las implementaciones del BASIC-80.

Ejemplo: CLOAD "MAX2"
Carga en memoria el archivo "M".

2.6 CLOSE

Formato: CLOSE [[#]⟨número archivo⟩ [,[#]⟨número archivo . . .⟩]]

Versión: Disco.

Propósito: Concluir la entrada/salida en un archivo en disco.

Observaciones: ⟨número archivo⟩ es el número bajo el cual el archivo fue abierto (OPEN). CLOSE sin argumentos cierra todos los archivos abiertos.

La asociación entre un archivo en particular y un número de archivo termina a la ejecución de una CLOSE. El archivo puede luego ser reabierto usando el mismo o un diferente número de archivo; de igual manera, ese número de archivo puede reutilizarse ahora para abrir (OPEN) cualquier archivo. Una CLOSE para un archivo de salida secuencial escribe el buffer final de salida.

La sentencia END y el comando NEW siempre cierran (CLOSE) automáticamente todos los archivos en disco. (STOP no cierra archivos en disco).

Ejemplo: Ver Apéndice B.

2.7 **COMMON**

Formato: COMMON ⟨lista de variables⟩

Versión: Disco.

Propósito: Transferir variables a un programa encadenado (CHAIN).

Observaciones: La sentencia COMMON se usa en conjunción con la sentencia CHAIN. Las sentencias COMMON pueden aparecer en cualquier lugar de un programa, aun cuando se recomienda que aparezcan al comienzo. En más de una sentencia COMMON no puede aparecer la misma variable. Las variables de matriz se especifican agregando "()" al nombre de la variable. Si han de transferirse todas las variables, use CHAIN con la opción ALL y omita la sentencia COMMON.

Ejemplo:
```
100 COMMON A,B,C,D(),G$
110 CHAIN "PROG3+,10
.
.
.
```

2.8 CONT

Formato: CONT

Versiones: 8K, Ampliado, Disco.

Propósito: Continuar la ejecución del programa después de digitarse Control-C o de haberse ejecutado una sentencia STOP o END.

Observaciones: La ejecución se reanuda en el punto en el que se produjo la interrupción. Si la interrupción se produjo después de una indicación de una sentencia INPUT, la ejecución continúa con la reimpresión de la indicación (? o serie de indicación).

CONT se usa por lo común en conjunción con STOP para corregir errores. Al parar la ejecución, pueden examinarse los valores intermedios y modificarlos usando sentencias en modalidad directa. La ejecución puede reanudarse con CONT o con GOTO en modalidad directa, que reanuda la ejecución en un número de línea especificado. Con las versiones Ampliado y Disco, CONT puede usarse para continuar la ejecución después de un error.

CONT no es válido si el programa ha sido editado durante la interrupción. En el BASIC-80 8K, no puede CONTinuarse la ejecución si durante la interrupción se ha producido un error en modalidad directa.

Ejemplo: Ver ejemplo en la Sección 2.61, STOP.

2.9. **CSAVE**

Formatos: CSAVE ⟨expresión de serie⟩

CSAVE* ⟨nombre variable matriz⟩

Versiones: 8K (cassette), Ampliado (cassette).

Propósito: Resguardar en cinta de cassette el programa o matriz corrientemente en memoria.

Observaciones: Todo programa o matriz resguardado en cinta se identifica con un nombrearchivo. Al ejecutarse el comando CSAVE ⟨expresión de serie⟩, BASIC-80 resguarda en cinta el programa corrientemente en memoria y usa el primer caracter de expresión de serie como nombrearchivo. ⟨expresión de serie⟩ puede tener más de un caracter, pero sólo se utiliza el primero como nombrearchivo.

Al ejecutarse el comando CSAVE* ⟨nombre variable matriz⟩, BASIC-80 resguarda en cinta la matriz especificada. La matriz debe ser numérica. Los elementos de una matriz multidimensional se resguardan con el subíndice de extrema izquierda que cambia más velozmente.

CSAVE puede usarse como una sentencia de programa o como un comando en modalidad directa. Antes de ejecutar una CSAVE o CSAVE*, compruebe que la grabadora de cassette esté adecuadamente conectada y en modalidad "Record".

Ver asimismo CLOAD, Sección 2.5.

NOTA: CSAVE y CLOAD no se incluyen en todas las implementaciones del BASIC-80.

Ejemplo: CSAVE "TIMER".
Resguarda el programa corrientemente en memoria, en cassette, bajo el nombrearchivo "T".

2.10 **DATA**

Formato: DATA ⟨lista de constantes⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Almacenar las constantes numéricas y de serie accedidas por la(s) sentencia(s) READ del programa. (Ver READ, Sección 2.54).

Observaciones: Las sentencias DATA no son ejecutables y pueden ubicarse en cualquier lugar del programa. Una sentencia DATA puede contener tantas constantes como quepan en una línea (separadas mediante comas), pudiendo usarse cualquier número de sentencias DATA en un programa. Las sentencias READ acceden las sentencias DATA en orden (por número de línea) y puede pensarse en los datos allí contenidos como en una lista continua de ítems, sea cual fuere la cantidad de éstos que hubiera en una línea o el lugar en el que las líneas se colocan en el programa.

⟨lista de constantes⟩ puede contener constantes numéricas de cualquier formato, esto es, punto fijo, punto flotante o enteras. (En la lista se permiten expresiones no numéricas). Las constantes de serie en las sentencias DATA deben estar entre comillas solamente si contienen comas, dos puntos o espacios significativos a la izquierda o derecha. De otra manera, no se necesitan las comillas.

El tipo de variable (numérica o de serie) dado en la sentencia READ debe concordar con la correspondiente constante en la sentencia DATA.

Las sentencias DATA pueden volver a leerse desde el comienzo mediante el uso de la sentencia RESTORE (Sección 2.57).

Ejemplo: Ver ejemplos en la Sección 2.54, READ.

2.11 **DEF FN**

Formato: DEF FN ⟨nombre⟩ [(⟨lista parámetros⟩)] = ⟨definición función⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Definir y nombrar una función que escribe el usuario.

Observaciones: ⟨nombre⟩ debe ser un nombre válido de variable. Este nombre, precedido por FN, se convierte en el nombre de la función. ⟨lista parámetros⟩ consta de aquellos nombres de variables de la definición de la función que han de ser reemplazados al llamarse la función. Los ítems de la lista se separan con comas. ⟨definición función⟩ es una expresión que realiza la operación de la función. Está limitada a una línea. Los nombres de variables que aparecen en esta expresión sirven sólo para definir la función; no afectan las variables de programa que tienen el mismo nombre. Un nombre de variable usado en una definición de función puede o no aparecer en la lista de parámetros. Si lo hace, al llamarse la función se suministra el valor del parámetro. En caso contrario, se usa el valor corriente de la variable.

Las variables de la lista de parámetros representan, en una base de una a una, las variables o valores de argumentos que se darán en el llamado de la función. (Recuerde, en la versión de 8K sólo se permite un argumento en un llamado de función; por lo tanto, la sentencia DEF FN contendrá sólo una variable).

En el BASIC-80 Ampliado y Disco, las funciones definidas por el usuario pueden ser numéricas o de serie; en 8K, no se permiten funciones de serie definidas por el usuario. Si en el nombre de la función se especificara un tipo, el valor de la expresión se fuerza a ese tipo antes de ser devuelto a la sentencia llamadora. Si se especificara un tipo en el nombre de la función y no concuerda el tipo de argumento, se produce un error de "Falta de concordancia de tipo".

Una sentencia DEF FN debe ejecutarse antes que pueda llamarse la función que define. Si se llamara una función antes de ser definida, se produce un error de "Función del usuario sin definir". DEF FN es inválida en la modalidad directa.

Ejemplo:

```
.
.
410 DEF FNAB (X,Y)=X^3/Y^2
420 T=FNAB(I,J)
.
.
```

La línea 410 define la función FNAB. La función se llama en la línea 420.

2.12 DEFINT/SNG/DBL/STR

Formato: DEF ⟨tipo⟩ ⟨límite(s) de letras⟩
en donde ⟨tipo⟩ es INT, SNG, DBL o STR.

Versiones: Ampliado, Disco.

Propósito: Declarar tipos de variables como entero, precisión simple, precisión doble o de serie.

Observaciones: Una sentencia DEFtipo declara que los nombres de las variables que comienzan con la(s) letra(s) especificada(s) serán de ese tipo. Sin embargo, un caracter de declaración de tipo tiene siempre procedencia sobre una sentencia DEFtipo en la digitación de una variable.

Si no se detectaran sentencias de declaración de tipo, el BASIC-80 presupone que todas las variables sin caracteres de declaración son de precisión simple.

```
10 DEFDBL L-P      Todas las variables que comienzan con las letras L, M, N, O y P serán variables de precisión doble.
10 DEFSTR A        Todas las variables que comienzan con la letra A serán variables de serie.
10 DEFINT I-N, WZ  Todas las variables que comienzan con las letras I, J, K, L, M, N, W, X, Y y Z serán variables de entero.
```

2.13 DEF USR

Formato: DEF USR [⟨dígito⟩] = ⟨expresión entera⟩

Versiones: Ampliado, Disco.

Propósito: Especificar la dirección inicial de una subrutina en lenguaje assembly.

Observaciones: ⟨dígito⟩ puede ser cualquier dígito de 0 a 9. El dígito corresponde al número de la rutina USR cuya dirección se está especificando. Si se omitiera ⟨dígito⟩, se presupone DEF USR0. El valor de ⟨expresión entera⟩ es la dirección inicial de la rutina USR. Ver Apéndice C, Subrutinas en Lenguaje Assembly.

En un programa puede aparecer cualquier número de sentencias DEF USR, para redefinir direcciones iniciales de subrutinas, permitiendo así acceder tantas subrutinas como fuere necesario.

Ejemplos:

```
.
.
.
200 DEF USR0=24000
210 X=USR0 (Y^2/2.89)
.
.
.
```

2.14 DELETE

Formato: DELETE [⟨número de línea⟩] [—⟨número de línea⟩]

Versiones: Ampliado, Disco.

Propósito: Eliminar líneas del programa.

Observaciones: BASIC-80 vuelve siempre a nivel de comando después de ejecutarse una DELETE. Si no existiera ⟨número de línea⟩, se produce un error "Llamado inválido de función".

Ejemplos:

```
DELETE 40      Elimina la línea 40.
DELETE 40-100  Elimina las líneas 40 a 100, inclusive.
DELETE 40      Elimina todas las líneas hasta e incluso la 40.
```

2.15 DIM

Formato: DIM ⟨lista de variables con subíndices⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Especificar los valores máximos de subíndices de variables de matriz y asignar almacenamiento en función de ello.

Observaciones: Si se usara un nombre de variable de matriz sin una sentencia DIM, se presupone que el valor máximo de su(s) subíndice(s) es 10. Si se usara un subíndice mayor que el máximo especificado se produce un error de "Subíndice fuera de límite". El valor mínimo de un subíndice es siempre 0, salvo que de otra manera se especificara con la sentencia OPTION BASE. (Ver Sección 2.46).

La sentencia DIM posiciona en un valor inicial de cero a todos los elementos de las matrices especificadas.

Ejemplo:

```
10 DIM A(20)
20 FOR I=0 TO 20
30 READ A(I)
40 NEXT I
.
.
.
```

2.16 EDIT

Formato: EDIT ⟨número de línea⟩

Versiones: Ampliado, Disco.

Propósito: Ingresar en Modalidad Edit en la línea especificada.

Observaciones: En Modalidad Edit, es posible editar porciones de una línea sin digitar nuevamente toda la línea. Al ingresar en la Modalidad Edit, el BASIC-80 digita el número de la línea a editar, luego un espacio y espera un subcomando en Modalidad Edit.

Subcomandos en Modalidad Edit

Los subcomandos de la modalidad Edit se usan para trasladar el cursor o para insertar, eliminar, reemplazar o buscar texto dentro de una línea. Los subcomandos no tienen eco. La mayoría de los subcomandos en la Modalidad Edit pueden ser precedidos por un entero que hace que el comando sea ejecutado ese número de veces. Cuando no se especifica ese entero, se presupone 1.

Los subcomandos de la Modalidad Edit pueden clasificarse de acuerdo con las siguientes funciones:

1. Movimiento del cursor
2. Inserción de texto
3. Eliminación de texto
4. Búsqueda de texto
5. Reemplazo de texto
6. Terminación y reanudación de la Modalidad Edit

NOTA: En las descripciones que siguen, ⟨ch⟩ representa cualquier caracter; ⟨texto⟩, una serie de caracteres de longitud arbitraria; [i], un entero opcional (el valor por defecto es 1), y $, la tecla Escape (o de modalidad alternada).

1. Movimiento del Cursor

Espaciado Use la barra espaciadora para trasladar el cursor a la derecha. [i] Espaciado traslada el cursor i espacios a la derecha. Los caracteres se imprimen a medida que espacía sobre ellos.

Rubout En Modalidad Edit, [i] Rubout traslada el cursor i espacios a la izquierda (retroceso). Los caracteres se imprimen a medida que retrocede sobre ellos.

2. Inserción de texto

I I⟨texto⟩ inserta el ⟨texto⟩ en la posición corriente del cursor. Los caracteres que se insertan se imprimen en la terminal. Para terminar la inserción, digite Escape. Si durante el comando Insert se digitara Retorno de Carro, el efecto es el mismo que digitar Escape y luego Retorno de Carro. Durante el comando Insert puede usarse la tecla Rubout o Delete de la terminal para eliminar los caracteres a la izquierda del cursor. Si se intentara insertar un caracter que hiciera a la línea más larga que 255 caracteres, se digita un timbre (Control-G) y el caracter no se imprime.

X — El subcomando X se usa para ampliar la línea. X traslada el cursor al final de la línea, entra en modalidad de inserción y permite la inserción de texto tal como si se hubiese emitido un comando Insert. Al terminar de ampliar la línea, digite Escape o Retorno de Carro.

3. Eliminación de texto

D — [i] D elimina i caracteres a la derecha del cursor. Los caracteres eliminados tienen eco entre barras inversas y el cursor se posiciona a la derecha del último caracter eliminado. Si a la derecha del cursor hubiera menos de i caracteres, iD elimina el resto de la línea.

H — H elimina todos los caracteres a la derecha del cursor y luego ingresa automáticamente en modalidad de inserción. H es útil para reemplazar sentencias al final de una línea.

4. Búsqueda de texto

S — El subcomando [i]S ⟨ch⟩ busca la iª ocurrencia de ⟨ch⟩ y posiciona el cursor delante de ella. En la búsqueda no se incluye el caracter que está en la posición corriente del cursor. Si no se hallara ⟨ch⟩, el cursor parará al final de la línea. Se imprimen todos los caracteres sobre los que se ha pasado durante la búsqueda.

K — El subcomando [i]K ⟨ch⟩ es similar a [i]S ⟨ch⟩, excepto que se eliminan todos los caracteres sobre los que se ha pasado durante la búsqueda. El cursor se posiciona delante de ⟨ch⟩ y los caracteres eliminados se encierran entre barras inversas.

5. Reemplazo de texto

C — El subcomando C ⟨ch⟩ modifica el siguiente caracter a ⟨ch⟩. Si deseara modificar los siguientes i caracteres, use el subcomando iC, seguido por i caracteres. Tras digitarse el iº nuevo caracter, se sale de la modalidad de modificación y volverá usted a la Modalidad Edit.

6. Terminación y reanudación de la Modalidad Edit

⟨cr⟩ — La digitación de Retorno de Carro imprime el resto de la línea, resguarda las modificaciones que usted hubiera hecho y sale de la Modalidad Edit.

E — El subcomando E tiene el mismo efecto que Retorno de Carro, salvo que no se imprime el resto de la línea.

Q — El subcomando Q vuelve a nivel de comando BASIC-80, **sin** resguardar ninguna de las modificaciones hechas en la línea durante la Modalidad Edit.

L — El subcomando L lista el resto de la línea (resguardando las modificaciones hechas hasta el momento) y reposiciona el cursor al comienzo de la línea, aún en Modalidad Edit. L se usa por lo común para listar la línea al ingresar en Modalidad Edit.

A — El subcomando A le permite editar nuevamente una línea. Restaura la línea original y reposiciona el cursor a su comienzo.

NOTA: Si el BASIC-80 recibiera un comando irreconocible o un caracter inválido mientras está en Modalidad Edit, imprime un timbre (Control-G) y se ignora el comando o caracter.

Errores de Sintaxis

Al detectarse un error de sintaxis durante la ejecución de un programa, el BASIC-80 ingresa automáticamente en Modalidad Edit en la línea que provocó el error. Por ejemplo:

```
10 K = 2(4)
RUN
?Syntax error in 10
10
```

Al terminar usted de editar la línea y digitar Retorno de Carro (o el subcomando E), el BASIC-80 reinserta la línea, lo que hace perder todos los valores de variables. Para conservarlos con miras a su examen, salga primero de la Modalidad Edit con el subcomando Q. El BASIC-80 retornará a nivel de comando, conservándose todos los valores variables.

Control-A

Para ingresar en Modalidad Edit en la línea que está digitando, digite Control-A. El BASIC-80 responde con un retorno de carro, un signo de exclamación (!) y un espacio. El cursor se posicionará en el primer caracter de la línea. Continúe, digitando un subcomando en Modalidad Edit.

NOTA: Recuerde, si acaba de ingresar una línea y desea retroceder y editarla, el comando "EDIT." ingresará en Modalidad Edit en la línea corriente. (El símbolo de número de línea "." se refiere siempre a la línea corriente).

2.17 **END**

Formato: END

Versiones: 8K, Ampliado, Disco.

Propósito: Terminar la ejecución del programa, cerrar todos los archivos y volver a nivel de comando.

Observaciones: Las sentencias END pueden colocarse en cualquier lugar del programa para terminar la ejecución. Contrariamente a la sentencia STOP, END no imprime un mensaje BREAK (interrupción). Es opcional una sentencia END al comienzo del programa. El BASIC-80 siempre vuelve a nivel de comando tras ejecutarse una END.

Ejemplo: 520 IF K>1000 THEN END ELSE GOTO 20

2.18 **ERASE**

Formato: ERASE ⟨lista variables de matriz⟩

Versiones: Ampliado, Disco.

Propósito: Eliminar matrices de un programa.

Observaciones: Las matrices pueden redimensionarse después de haber sido ERASE (borradas) o bien puede usarse con otro propósito el espacio de matriz previamente asignado. Si se intentara redimensionar una matriz sin primero borrarla se produce un error de "Matriz redimensionada".

NOTA: El compilador BASIC de Microsoft no apoya ERASE.

Ejemplo:

```
.
.
.
450 ERASE A,B
460 DIM B(99)
.
.
.
```

2.19 **VARIABLES ERR y ERL**

Al ingresarse en una subrutina de atención de errores, la variable ERR contiene el código de error y la variable ERL el número de la línea en la que se detectó el error. Las variables ERR y ERL se usan por lo común en sentencias IF...THEN para dirigir el flujo del programa en la rutina de desvío de errores.

Si la sentencia que provocó el error fuera una de modalidad directa, ERL contendrá 65535. Para probar si se ha producido un error en una sentencia directa, use IF 65535 = ERL THEN...

En caso contrario use:

IF ERR = código de error THEN...

IF ERL = número de línea THEN...

Si el número de línea no estuviera en el costado derecho del operador relacional, no puede renumerarse con RENUM. Puesto que ERL y ERR son variables reservadas, no pueden aparecer a la izquierda del signo igual de una sentencia LET (asignación). En el Anexo F se listan los códigos de error del BASIC-80.

2.20 **ERROR**

Formato: ERROR ⟨expresión entera⟩

Versiones: Ampliado, Disco.

Propósito: 1) Simular la ocurrencia de un error BASIC-80; ó 2) permitir que el usuario defina los códigos de error.

Observaciones: El valor de expresión entera debe ser mayor que 0 y menor que 255. Si el valor de ⟨expresión entera⟩ fuese igual a un código de error que ya usa el BASIC-80 (ver Anexo F), la sentencia ERROR simulará la ocurrencia de ese error, imprimiéndose el correspondiente mensaje de error. (Ver ejemplo 1.) Para definir su propio código de error, use un valor que sea mayor que cualquier otro que utilice

el BASIC-80. (Es preferible usar los valores más altos que se dispongan, a fin de mantener compatibilidad cuando se agreguen más códigos de error al BASIC-80.) Este código de error definido por el usuario puede luego ser convenientemente atendido en una rutina de desvío por error. (Ver ejemplo 2).

Si una sentencia ERROR especificara un código para el que no se ha definido ningún mensaje de error, el BASIC-80 responde con el mensaje UNPRINTABLE ERROR (error imposible de imprimir). La ejecución de una sentencia ERROR para la cual no exista una rutina de desvío imprime un mensaje de error y detiene la ejecución.

Ejemplo 1:

```
LIST
10 S = 10
20 T = 5
30 ERROR S + T
40 END
Ok
Run
String too long in line 30 (Serie demasiado larga en línea 30)
0, en modalidad directa:
Ok
ERROR 15          (usted digita esta línea)
String too long   (BASIC-80 digita esta línea)
Ok
```

Ejemplo 2:

```
.
.
.
110 ON ERROR GOTO 400
120 INPUT "CUAL ES SU APUESTA";B
130 SI B>5000 THEN ERROR 210
.
.
.
400 IF ERR = 210 THEN PRINT "EL LIMITE DE LA CASA ES $ 5000"
410 IR ERL = 130 THEN RESUME 120
.
.
.
```

2.21 **FIELD**

Formato: FIELD [#] ⟨número archivo⟩, ⟨ancho campo⟩ AS ⟨variable de serie⟩...

Versión: Disco.

Propósito: Asignar espacio para variables en un buffer de archivo al azar.

Observaciones: Para extraer datos de un buffer al azar después de una GET o para ingresar datos antes de una PUT, debe haberse ejecutado una sentencia FIELD.

⟨número archivo⟩ es el número con el cual el archivo fue abierto (OPEN). ⟨ancho archivo⟩ es el número de caracteres a asignar a ⟨variable de serie⟩. Por ejemplo,

FIELD 1, 20 AS N$, 10 AS ID$, 40 AS ADD$

asigna las primeras 20 posiciones (bytes) del buffer de archivo al azar a la variable de serie N$, las siguientes 10 posiciones a ID$, y las siguientes 40 a ADD$. FIELD **no** coloca datos en el buffer de archivos al azar. (Ver LSET/RSET y GET).

El número total de bytes asignados en una sentencia FIELD no debe exceder la longitud de registro especificada al abrirse el archivo. En caso contrario se produce un error de "Overflow de campo". (La longitud por defecto del registro es 128).

Para el mismo archivo puede ejecutarse cualquier número de sentencias FIELD, y todas las sentencias FIELD que hayan sido ejecutadas están en vigencia al mismo tiempo.

Ejemplo: Ver Anexo B.

NOTA: No use un nombre de variable que haya sido objeto de FIELD en una sentencia INPUT o LET. Una vez que el nombre de una variable haya sido objeto de FIELD, apunta a la ubicación correcta del buffer de archivo al azar. De ejecutarse una subsiguiente sentencia INPUT o LET con ese nombre de variable, el indicador ('pointer') de la variable se transfiere a espacio de serie.

2.22 FOR . . . NEXT

Formato: FOR ⟨variable⟩=x TO y [STEP z]

.
.
.

NEXT [⟨variable⟩] [,⟨variable⟩ . . .]

en donde x, y y z son expresiones numéricas.

Versiones: 8K, Ampliado, Disco.

Propósito: Permitir que una serie de instrucciones se cumplan en un lazo un número dado de veces.

Observaciones: ⟨variable⟩ se usa como contador. La primera expresión numérica (x) es el valor inicial del contador. La segunda expresión numérica (y) es el valor final del contador. Las líneas del programa que siguen a la sentencia FOR se ejecutan hasta detectarse la sentencia NEXT. Luego, el contador se incrementa en el total especificado por STEP. Se verifica si el valor del contador es ahora mayor que el valor final (y). Si no fuese mayor, el BASIC-80 bifurca a la sentencia que sigue a la sentencia FOR y el proceso se repite. Si fuese mayor, la ejecución continúa con la sentencia que sigue a la sentencia NEXT. Este es un lazo FOR . . . NEXT. Si no se especificara STEP, se presupone que el incremento es uno. Si STEP fuese negativo, el valor final del contador se posiciona de modo que sea menor que el valor inicial. El contador se decrementa en cada una de las ocasiones a través del lazo, y éste se ejecuta hasta que el contador sea menor que el valor final.

Se omite el cuerpo del lazo si el valor inicial del lazo por el signo del paso excede el valor final por el signo del paso.

Lazos Anidados

Los lazos FOR . . . NEXT pueden anidarse, esto es, un lazo FOR . . . NEXT puede colocarse dentro del contexto de otro lazo FOR . . . NEXT. Cuando se anidan lazos, cada uno de ellos ha de tener como contador un nombre único de variable. La sentencia NEXT para el lazo interior debe aparecer delante de la del lazo exterior. Si los lazos anidados tuvieran el mismo punto final, puede usarse una sola sentencia NEXT para todos ellos.

Pueden omitirse la(s) variable(s) de la sentencia NEXT, en cuyo caso la sentencia NEXT concordará con la más reciente sentencia FOR. Si se detectara una sentencia NEXT antes que su correspondiente sentencia FOR, se emite un mensaje "NEXT sin FOR" y la ejecución termina.

Ejemplo 1:

```
10 K=10
20 FOR I=1 TO K STEP 2
30 PRINT I;
40 K=K+10
50 PRINT K
60 NEXT
RUN
  1  20
  3  30
  5  40
  7  50
  9  60
Ok
```

Ejemplo 2:

```
10 J=O
20 FOR I=1 TO J
30 PRINT I
40 NEXT I
```

En este ejemplo el lazo no se ejecuta puesto que el valor inicial del lazo excede al valor final.

Ejemplo 3:

```
20 FOR I=1 to I+5
10 I=5
30 PRINT I;
40 NEXT
RUN
 1  2  3  4  5  6  7  8  9  10
Ok
```

En este ejemplo el lazo se ejecuta diez veces. El valor final de la variable del lazo se posiciona siempre antes que se posiciona el valor final. (NOTA: Las versiones anteriores del BASIC-80 posicionan el valor inicial de la variable del lazo antes de posicionar el valor final; esto es, el lazo anterior se ejecutaría seis veces).

2.23 **GET**

Formato: GET [#] ⟨número archivo⟩ [,⟨número registro⟩]

Versión: Disco.

Propósito: Leer un registro desde un archivo en disco al azar en un buffer al azar.

Observaciones: ⟨número archivo⟩ es el número con el cual fue abierto (OPEN) el archivo. Si se omitiera ⟨número registro⟩, se lee en el buffer el siguiente registro (después de la última GET). El número mayor posible de registro es 32767.

Ejemplo: Ver Anexo B.

2.24 **GOSUB . . . RETURN**

Formato:

```
GOSUB ⟨número de línea⟩
.
.
.
RETURN
```

Versiones: 8K, Ampliado, Disco.

Propósito: Bifurcar a una subrutina y volver de ella.

Observaciones: ⟨número de línea⟩ es la primera línea de la subrutina.

Puede llamarse una subrutina cualquier número de veces en un programa, pudiendo asimismo llamarse una subrutina desde otra subrutina. La memoria de que se disponga es la única limitación de tal anidamiento de subrutinas.

La(s) sentencia(s) RETURN en una subrutina hace(n) que el BASIC-80 bifurque a la sentencia que sigue a la sentencia GOSUB más reciente. Una subrutina puede contener más de una sentencia RETURN, de imponer la lógica un retorno a diferentes puntos de la subrutina. Las subrutinas pueden aparecer en cualquier lugar del programa, pero se recomienda que puedan distinguirse rápidamente del programa principal. Para impedir una entrada involuntaria en la subrutina, puede ésta estar precedida por una sentencia STOP, END o GOTO que dirija el control del programa en torno a la subrutina.

Ejemplo:

```
10 GOSUB 40
20 PRINT "REGRESO DE LA SUBRUTINA"
30 END
40 PRINT "SUBRUTINA";
50 PRINT "EN";
```

```
60 PRINT "TRAMITE"
70 RETURN
RUN
SUBRUTINA EN TRAMITE
REGRESO DE LA SUBRUTINA
Ok
```

2.25 **GOTO**

Formato: GOTO ⟨número de línea⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Bifurcar incondicionalmente fuera de la secuencia normal del programa a un número de línea especificado.

Observaciones: Si ⟨número de línea⟩ fuese una sentencia ejecutable, se ejecuta esa sentencia y las que le siguen. Si no fuese ejecutable, la ejecución continúa en la primera sentencia ejecutable que se detecte después de ⟨número de línea⟩.

Ejemplo:

```
LIST
10 READ R
20 PRINT "R = ";R,
30 A = 3.14*R∧2
40 PRINT "AREA = ";A
50 GOTO 10
60 DATA 5,7,12
Ok
RUN
R = 5          AREA = 78.5
R = 7          AREA = 153.86
R = 12         AREA = 452.16
?Out of data in 10
Ok
```

2.26 **IF . . . THEN** [. . . **ELSE**] E **IF . . . GOTO**

Formato: IF ⟨expresión⟩ THEN ⟨sentencia(s)⟩ | ⟨número de línea⟩
[ELSE ⟨sentencia(s)⟩ | ⟨número de línea⟩]

Formato: IF ⟨expresión⟩ GOTO ⟨número de línea⟩
[ELSE ⟨sentencia(s)⟩ | ⟨número de línea⟩]

Versiones: 8K, Ampliado, Disco.

NOTA: La cláusula ELSE está permitida sólo en las versiones Ampliado y Disco.

Propósito: Tomar una decisión concerniente al flujo del programa en base al resultado devuelto por una expresión.

Observaciones: Si el resultado de ⟨expresión⟩ no fuese cero, se ejecuta la cláusula THEN o GOTO. THEN puede ser seguida por un número de línea para bifurcar a ella o por una o más sentencias a ejecutar. GOTO es siempre seguida por un número de línea. Si el resultado de ⟨expresión⟩ fuese cero, se ignora la clásula THEN o GOTO y se ejecuta la ELSE, de estar presente. La ejecución continúa con la siguiente sentencia ejecutable. (ELSE está permitida solamente en las versiones Ampliado y Disco.) Las versiones Ampliado y Disco permiten una coma delante de THEN.

Anidamiento de Sentencias IF

Las sentencias IF . . . THEN . . . ELSE pueden anidarse en las versiones Ampliado y Disco. El anidamiento está sólo limitado por la longitud de la línea. Por ejemplo:

```
IF X>Y THEN PRINT "MAYOR" ELSE IF Y>X
```

THEN PRINT "MENOR QUE" ELSE PRINT "IGUAL"

es una sentencia válida. Si la sentencia no contuviera el mismo número de cláusulas ELSE y THEN, toda ELSE concuerda con la THEN más próxima sin concordar. Por ejemplo:

IF A=B THEN IF B=C THEN PRINT "A=C"
ELSE PRINT "A<>C"

no imprimirá "A<>C" cuando A<>B.

Si una sentencia IF . . . THEN fuese seguida por un número de línea en modalidad directa, resulta un error de "Línea indefinida", salvo que previamente se hubiera ingresado, en modalidad directa, una sentencia con el número de línea especificado.

NOTA: Al usar IF para probar la igualdad de un valor que es resultado de un cálculo en punto flotante recuerde que la representación interna del valor puede no ser exacta. Por lo tanto, la prueba se hará con el límite en el cual puede variar la precisión del valor. Por ejemplo, para probar una variable calculada A contra el valor 1.0, use:

IF ABS (A—1.0)<1.0E—6 THEN . . .

Esta prueba resulta verdadera si el valor de A es 1.0, con un error relativo de menor de 1.0E—6.

Ejemplo 1: 200 IF I THEN GET#1,I

Esta sentencia GET el número de registro I si I no es cero.

Ejemplo 2: 100 IF (I<20)*(I>10) THEN DB=1979—1:GOTO 300
110 PRINT "FUERA DE LIMITE"
.
.
.

En este ejemplo, una prueba determina si I es mayor que 10 y menor que 20. Si I está dentro de estos límites, se calcula DB y la ejecución bifurca a la línea 300. Si I no está dentro de estos límites, la ejecución continúa con la línea 110.

Ejemplo 3: 210 IF IOFLAG THEN PRINT A$ ELSE LPRINT A$

Esta sentencia dirige la salida impresa a la terminal o a la impresora de líneas, en función del valor de una variable (IOFLAG). Si IOFLAG fuese cero, la salida se dirige a la impresora de líneas; en caso contrario, a la terminal.

2.27 **INPUT**

Formato: INPUT [;] [("serie de indicación");] ⟨lista de variables⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Permitir entrada desde la terminal durante la ejecución del programa.

Observaciones: Al detectarse una sentencia INPUT, la ejecución del programa entra en una pausa y se imprime un signo de interrogación para indicar que el programa está esperando datos. Si se incluyera ⟨"serie de indicación"⟩, la serie se imprime delante del signo de interrogación. Los datos requeridos se ingresan luego en la terminal.

Si INPUT fuera seguida de inmediato por un punto y coma, el retorno de carro que digita el usuario para ingresar datos no da como eco una secuencia retorno de carro/alimentación de línea.

Los datos que se ingresan se asignan a la(s) variable(s) dadas en ⟨lista de variables⟩. El número de ítems de datos provistos ha de ser el mismo que el número de variables de la lista. Los ítems de datos se separan con comas.

Los nombres de variables de la lista pueden ser de variables numéricas o de serie (incluso variables con subíndices). El tipo de todo ítem de datos al que se da entrada debe concordar con el especificado por el nombre de la variable. (No es necesario encerrar entre comillas las series entradas en una sentencia INPUT).

De responderse a INPUT con demasiados o pocos ítems, o con un tipo erróneo de valor (numérico en lugar de serie, etc.) se imprime el mensaje "?Redo from start" (Rehacer desde el comienzo?). No se asignan valores de entrada hasta que no se dé una respuesta aceptable.

En la versión 8K, INPUT no es válido en modalidad directa.

Ejemplos: 10 INPUT X
20 PRINT X "AL CUADRADO ES" X∧2

```
30 END
RUN
? 5  (el 5 ha sido digitado por el usuario en respuesta al signo de interrogación)
 5 AL CUADRADO ES 25
Ok
LIST
10 PI=3.14
20 INPUT "CUAL ES EL RADIO" ;R
30 A=PI*R/\2
40 PRINT "EL AREA DEL CIRCULO ES" ;A
50 PRINT
60 GOTO 20
Ok
RUN
CUAL ES EL RADIO? 7.4 (el usuario digita 7.4)
EL AREA DEL CIRCULO ES 171.946
CUAL ES EL RADIO?
etc.
```

2.28 **INPUT#**

Formato: INPUT# ⟨número archivo⟩, ⟨lista de variables⟩

Versión: Disco.

Propósito: Leer ítems de datos desde un archivo secuencial en disco y asignarlos a variables de programa.

Observaciones: ⟨número archivo⟩ es el número usado al abrirse el archivo para entrada. ⟨lista de variables⟩ contiene los nombres de variables que se asignarán a los ítems del archivo. (El tipo de variable debe concordar con el especificado por el nombre de la variable.) Con INPUT#, no se imprime signo de interrogación como en INPUT.

Los ítems de datos del archivo deberán aparecer tal como lo harían como si se estuvieran digitando en respuesta a una sentencia INPUT. Con valores numéricos, se ignoran los espacios a la izquierda, los retornos de carro y las alimentaciones de línea. El primer caracter que se detecta que no sea un espacio, retorno de carro o alimentación de línea se presupone que es el comienzo de un número. El número termina en un espacio, retorno de carro, alimentación de línea o coma.

Si el BASIC-80 estuviera explorando el archivo secuencial en disco por un ítem de serie, también se ignoran los espacios a la izquierda, los retornos de carro y las alimentaciones de línea. El primer caracter detectado que no sea un espacio, retorno de carro o alimentación de línea se presupone el comienzo de un ítem de serie. Si este primer caracter fuese comillas ("), el ítem de serie consistirá en todos los caracteres leídos entre la primera y la segunda comillas. Así, una serie entre comillas no puede contener comillas como caracter. Si el primer caracter de la serie no fuese comillas, la serie es sin comillas y terminará en una coma, retorno de carro o alimentación de línea (o después de haberse leído 255 caracteres). Si se alcanzara fin de archivo mientras se hace el INPUT de un ítem numérico o de serie, el ítem termina.

Ejemplo: Ver Anexo B.

2.29 **KILL**

Formato: KILL ⟨nombrearchivo⟩

Versión: Disco.

Propósito: Eliminar un archivo del disco.

Observaciones: Si se emitiera una sentencia KILL para un archivo actualmente abierto, se produce un error "Archivo ya abierto".

KILL se usa para todos los tipos de archivos en disco: de programas, de datos al azar y secuenciales de datos.

Ejemplo: 200 KILL "DATA1"

Ver asimismo Anexo B.

2.30 **LET**

Formato: [LET] ⟨variable⟩ = ⟨expresión⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Asignar a una variable el valor de una expresión.

Observaciones: Adviértase que la palabra LET es opcional; esto es, basta el signo igual al asignar una expresión al nombre de una variable.

Ejemplo:

```
110 LET D=12
120 LET E=12^2
130 LET F=12^4
140 LET SUM=D+E+F
.
.
.
ó
110 D=12
120 E=12^2
130 F=12^4
140 SUM=D+E+F
.
.
.
```

2.31 **LINE INPUT**

Formato: LINE INPUT [;] [⟨"serie de indicación"⟩;] ⟨variable de serie⟩

Versiones: Ampliado, Disco.

Propósito: Ingresar una línea completa (hasta 254 caracteres) en una variable de serie, sin el uso de delimitadores.

Observaciones: La serie de indicación en un literal de serie que se imprime en la terminal antes de aceptarse la entrada. No se imprime un signo de interrogación salvo que sea parte de la serie de indicación. Toda la entrada desde el final de la indicación al retorno de carro se asigna a ⟨variable de serie⟩.

Si LINE INPUT fuese seguido de inmediato por un punto y coma, el retorno de carro que digita el usuario para terminar la línea de entrada no da como eco una secuencia retorno de carro/alimentación de línea en la terminal.

Puede escaparse de una LINE INPUT digitando Control-C. El BASIC-80 retornará a nivel de comando y digitará Ok. De digitarse CONT, se reanuda la ejecución en la LINE INPUT.

Ejemplo: Ver ejemplo, Sección 2.32, LINE INPUT#.

2.32 **LINE INPUT#**

Formato: LINE INPUT# ⟨número archivo⟩, ⟨variable de serie⟩

Versión: Disco.

Propósito: Leer una línea completa (hasta 254 caracteres), sin delimitadores, desde un archivo secuencial de datos en disco en una variable de serie.

Observaciones: ⟨número archivo⟩ es el número con el que se abrió el archivo. ⟨variable de serie⟩ es el nombre de la variable a la cual se asignará la línea. LINE INPUT# lee todos los caracteres del archivo secuencial hasta un retorno de carro. Luego salta la secuencia retorno de carro/alimentación de línea y la siguiene LINE INPUT# lee todos los caracteres hasta el siguiente retorno de carro. (Si se detectara una secuencia alimentación de línea/retorno de carro, se conserva.)

LINE INPUT# es especialmente útil si cada una de las líneas de un archivo de datos se ha descompuesto en campos, o si un programa está leyendo un programa BASIC-80 resguardado en modalidad ASCII, como datos.

Ejemplo:

```
10 OPEN "0", 1, "LIST"
20 LINE INPUT "INFORMACION CLIENTE?"; C$
30 PRINT #1, C$
40 CLOSE 1
50 OPEN "I", 1, "LIST"
60 LINE INPUT#1, C$
70 PRINT C$
80 CLOSE 1
RUN
INFORMACION CLIENTE? LINDA JONES  234,4  MEMPHIS
LINDA JONES  234,4  MEMPHIS
Ok
```

2.33 LIST

Formato 1: LIST [⟨número de línea⟩]

Versiones: 8K, Ampliado, Disco.

Formato 2: LIST [⟨número de línea⟩ [—[⟨número de línea⟩]]]

Versiones: Ampliado, Disco.

Propósito: Listar por terminal el programa corrientemente en memoria, total o parcialmente.

Observaciones: El BASIC-80 vuelve siempre a nivel de comando después de ejecutarse LIST.

Formato 1: Si se omitiera ⟨número de línea⟩, se lista el programa comenzando en el número más bajo de línea. (El listado termina por fin del programa o al digitarse Control-C.) Si se incluyera ⟨número de línea⟩, la versión 8K listará el programa comenzando en esa línea; las versiones Ampliado y Disco listarán solamente la línea especificada.

Formato 2: Este formato permite las siguientes opciones:

1. Si sólo se especifica el primer número, se listan esa línea y todas las líneas de números más altos.
2. Si sólo se especifica el segundo número, se listan todas las líneas desde el comienzo del programa hasta esa línea.
3. Si se especifican ambos números, se lista todo el límite.

Ejemplos:

Formato 1:

LIST — Lista el programa que está corrientemente en la memoria.

LIST 500 — En la versión 8K, lista todas las líneas del programa desde la 500 hasta el final.
En las versiones Ampliado y Disco, lista la línea 500.

Formato 2:

LIST 150- — Lista todas las líneas desde la 150 hasta el final.

LIST -1000 — Lista todas las líneas desde la de número más bajo a la 1000.

LIST 150-1000 — Lista las líneas 150 a 1000, inclusive.

2.34 LLIST

Formato: LLIST [⟨número de línea⟩ [—[⟨número de línea⟩]]]

Versiones: Ampliado, Disco.

Propósito: Listar por la impresora de líneas el programa corrientemente en la memoria, total o parcialmente.

Observaciones: LLIST presupone una impresora de 132 caracteres.
El BASIC-80 siempre vuelve a nivel de comando después de ejecutarse LLIST. Las opciones de LLIST son las mismas que las de LIST, Formato 2.

NOTA: LLIST y LPRINT no se incluyen en todas las implementaciones del BASIC-80.

Ejemplo: Ver los ejemplos LIST, Formato 2.

2.35 **LOAD**

Formato: LOAD ⟨nombrearchivo⟩ [,R]

Versión: Disco.

Propósito: Cargar un archivo en memoria desde disco.

Observaciones: ⟨nombrearchivo⟩ es el nombre utilizado al resguardarse (SAVE) el archivo. (Con CP/M, se provee la extensión por defecto .BAS).

LOAD cierra todos los archivos abiertos y elimina todas las variables y líneas del programa que residen corrientemente en la memoria antes de cargar el programa designado. Sin embargo, si con LOAD se usara la opción "R", el programa se ejecuta (RUN) después de ser cargado (LOAD) y se mantienen abiertos todos los archivos de datos que están abiertos. Así, puede usarse LOAD con la opción "R" para encadenar varios programas (o segmentos del mismo programa). Puede transferirse información entre los programas usando sus archivos de datos en disco.

Ejemplo: LOAD "STRTRK",R

2.36 **LPRINT y LPRINT USING**

Formato: LPRINT [⟨lista de expresiones⟩]

LPRINT USING ⟨expr serie⟩;⟨lista de expresiones⟩

Versiones: Ampliado, Disco.

Propósito: Imprimir datos en la impresora de líneas.

Observaciones: Igual que PRINT y PRINT USING, salvo que la salida se dirige a la impresora de líneas. Ver Secciones 2.49 y 2.50.

LPRINT presupone una impresora de 132 caracteres.

NOTA: LPRINT y LLIST no se incluyen en todas las implementaciones del BASIC-80.

2.37 **LSET y RSET**

Formato: LSET ⟨variable de serie⟩ = ⟨expresión de serie⟩

RSET ⟨variable de serie⟩ = ⟨expresión de serie⟩

Versión: Disco.

Propósito: Transferir datos de la memoria a un buffer de archivo al azar (en preparación de una sentencia PUT).

Observaciones: Si ⟨expresión de serie⟩ requiriera menos bytes que los que fueron FIELD en ⟨variable de serie⟩, LSET justifica a la izquierda la serie en el campo y RSET a la derecha. (Para rellenar las posiciones adicionales se usan espacios). Si la serie fuera demasiado larga para el campo, se eliminan caracteres a la derecha. Los valores numéricos deben convertirse a series antes de ser LSET o RSET. (Ver las funciones MKI$, MKS$ y MKD$, Sección 3.25).

Ejemplos:
```
150 LSET A$=MKS$(AMT)
160 LSET D$=DESC($)
```
Ver asimismo Anexo B.

NOTA: LSET o RSET pueden también usarse con una variable de serie que no ha sido objeto de FIELD para justificar a la izquierda o a la derecha una serie en un campo dado. Por ejemplo, las líneas de programa

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

justifican a la derecha la serie N$ en un campo de 20 caracteres. Esto puede ser muy práctico para dar formato a salida impresa.

2.38 **MERGE**

Formato: MERGE ⟨nombrearchivo⟩

Versión: Disco.

Propósito: Intercalar un archivo en disco especificado en el programa que está corrientemente en memoria.

Observaciones: ⟨nombrearchivo⟩ es el nombre usado al resguardarse (SAVE) el archivo. (Con CP/M, se provee la extensión por defecto .BAS). El archivo debe haber sido resguardado en formato ASCII. (De no ser así, se produce un error "Modalidad errónea de archivo").

Si cualesquiera líneas del archivo en disco tuvieran los mismos números de línea que las líneas del programa que está en la memoria, las líneas del archivo en disco reemplazarán a las correspondientes que están en la memoria. (Puede pensarse en MERGE como en la "inserción" de líneas del programa en disco en el programa que está en la memoria).

El BASIC-80 vuelve siempre a nivel de comando después de ejecutar un comando MERGE.

Ejemplo: MERGE "NUMBRS"

2.39 **MID$**

Formato: MID$ (⟨expr serie1⟩ ,n[,m])=⟨expr serie2⟩
en donde n y m son expresiones enteras y ⟨expr serie1⟩ y ⟨expr serie2⟩ son expresiones de serie.

Versiones: Ampliado, Disco.

Propósito: Reemplazar una porción de una serie con otra serie.

Observaciones: Los caracteres de ⟨expr serie1⟩, comenzando en la posición n, son reemplazados por los caracteres de ⟨expr serie2⟩. La m opcional se refiere al número de caracteres de ⟨expr serie2⟩ que se usarán en el reemplazo. Si se omitiera m, se usa toda la ⟨expr serie2⟩. Sin embargo, se omita o incluya m, el reemplazo de los caracteres nunca va más allá de la longitud original de ⟨expr serie1⟩.

Ejemplo:
```
10 A$="KANSAS CITY, MO"
20 MID$(A$,14)="KS"
30 PRINT A$
RUN
KANSAS CITY, KS
```
Asimismo puede usarse MID$ como una función que devuelve una subserie de una serie dada. (Ver Sección 3.24.)

2.40 **NAME**

Formato: NAME ⟨nombrearchivo anterior⟩ AS ⟨nombrearchivo nuevo⟩

Versión: Disco.

Propósito: Cambiar el nombre de un archivo en disco.

Observaciones: ⟨nombrearchivo anterior⟩ debe existir y ⟨nombrearchivo nuevo⟩ no. De no ser así resultará un error. Después de un comando NAME, el archivo existe en el mismo disco, en la misma área de espacio en disco, con el nuevo nombre.

Ejemplo:
```
Ok
NAME "ACCTS" AS "LEDGER"
Ok
```
En este ejemplo el archivo que antes se llamaba ACCTS se llama ahora LEDGER.

2.41 **NEW**

Formato: NEW

Versiones: 8K, Ampliado, Disco.

Propósito: Eliminar el programa que está corrientemente en la memoria y borrar todas las variables.

Observaciones: NEW se ingresa a nivel de comando para borrar la memoria antes de ingresar un nuevo programa.

El BASIC-80 vuelve siempre a nivel de comando después de ejecutarse una NEW.

2.42 **NULL**

Formato: NULL ⟨expresión entera⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Fijar el número de nulos que han de imprimirse al final de cada línea.

Observaciones: Para perforaciones en cinta de 10 caracteres por segundo, ⟨expresión entera⟩ debe ser >=3. Cuando no se perforan cintas, ⟨expresión entera⟩ debe ser 0 ó 1 para teletipos o CRTs compatibles con teletipo. ⟨expresión entera⟩ será 2 ó 3 para impresoras de 30 cps. El valor por defecto es 0.

Ejemplo:
```
Ok
NULL 2
Ok
100 INPUT X
200 IF X<50 GOTO 800
.
.
.
```
Después de cada línea se imprimirán dos caracteres nulos.

2.43 **ON ERROR GOTO**

Formato: ON ERROR GOTO ⟨número de línea⟩

Versiones: Ampliado, Disco.

Propósito: Permitir el desvío por error y especificar la primera línea de la subrutina de atención de errores.

Observaciones: Una vez activado el desvío por errores, todos los errores detectados, incluso los de modalidad directa (por ejemplo, errores de sintaxis) provocarán un salto a la subrutina de atención de errores especificada. Si no existiera ⟨número de línea⟩, resulta un error de "Línea no definida". Para desactivar el desvío por errores ejecute una ON ERROR GOTO 0. Los errores subsiguientes imprimirán un mensaje de error y se detendrá la ejecución. Una sentencia ON ERROR GOTO 0 que aparezca en una subrutina de desvío por error hace que el BASIC-80 se detenga e imprima el mensaje de error correspondiente al error que ha provocado el desvío. Se recomienda que todas las subrutinas de desvío por error ejecutan una sentencia ON ERROR GOTO 0 si se detectara un error para el que no existe una acción de recuperación.

NOTA: Si se produjera un error durante la ejecución de una subrutina de atención de errores, se imprime el mensaje de error BASIC y la ejecución termina. El desvío por error no se produce dentro de la subrutina de atención de errores.

Ejemplo: 10 ON ERROR GOTO 1000

2.44 **ON . . . GOSUB** y **ON . . . GOTO**

Formato: ON ⟨expresión⟩ GOTO ⟨lista de números de línea⟩
ON ⟨expresión⟩ GOSUB ⟨lista de números de línea⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Bifurcar a uno de varios números de línea especificados, en función del valor devuelto al evaluarse una expresión.

Observaciones: El valor de ⟨expresión⟩ determina el número de línea de la lista que se usará en la bifurcación. Por ejemplo, si el valor fuese tres, el tercer número de línea de la lista será el destino de la bifurcación. (Si el valor no fuese entero, se redondea la porción fraccionaria).

En la sentencia ON . . . GOSUB cada uno de los números de línea de la lista debe ser el número de la primera línea de una subrutina.

Si el valor de ⟨expresión⟩ fuese cero o mayor que el número de ítems de la lista (pero menor que o igual a 255), el BASIC continúa con la siguiente sentencia ejecutable. Si el valor de ⟨expresión⟩ fuese negativo o mayor que 255, se produce un error "Llamado inválido de función".

Ejemplo: 100 ON L-1 GOTO 150,300,320,390

2.45 **OPEN**

Formato: OPEN ⟨modalidad⟩, [#] ⟨número archivo⟩, ⟨nombrearchivo⟩, [⟨long reg⟩]

Versión: Disco.

Propósito: Permitir entrada/salida en un archivo en disco.

Observaciones: Un archivo en disco debe abrirse (OPEN) antes de que en él pueda ejecutarse una operación de entrada/salida. OPEN asigna un buffer para la entrada/salida del archivo y determina la modalidad de acceso que se usará con el buffer.

⟨modalidad⟩ es una expresión de serie cuyo primer caracter es uno de los siguientes:

O especifica modalidad secuencial de salida

I especifica modalidad secuencial de entrada

R especifica modalidad al azar de entrada/salida

⟨número archivo⟩ es una expresión entera cuyo valor está entre uno y quince. El número se asocia luego con el archivo mientras éste esté abierto y se utiliza para referir al archivo otras sentencias de entrada/salida en disco.

⟨nombrearchivo⟩ es una expresión de serie que contiene un nombre que se conforma a su sistema operativo en cuanto a nombres de archivos en disco.

⟨long reg⟩ es una expresión entera que, de incluirse, fija la longitud del registro para archivos al azar. La longitud por defecto del registro es de 128 bytes. (Ver asimismo página A-3).

NOTA: Un archivo puede abrirse para entrada secuencial o acceso al azar con más de un número de archivo a la vez. Sin embargo, un archivo puede abrirse para salida sólo con un número de archivo a la vez.

Ejemplo: 10 OPEN "I",2,"INVEN"

Ver asimismo Anexo B.

2.46 **OPTION BASE**

Formato: OPTION BASE n

Versiones: 8K, Ampliado, Disco.

Propósito: Declarar el valor mínimo de subíndices de matriz.

Observaciones: La base por defecto es 0. Si se ejecuta la sentencia

OPTION BASE 1

el valor más bajo que puede tener el subíndice de matriz es 1.

2.47 **OUT**

Formato: OUT I,J

en donde I y J son expresiones enteras entre los límites 0 a 255.

Versiones: 8K, Ampliado, Disco.

Propósito: Transmitir un byte a una vía de salida de máquina.

Observaciones: La expresión entera I es el número de vía; J, los datos a transmitir.

Ejemplo: 100 OUT 32,100

2.48 **POKE**

Formato: POKE I,J

en donde I y J son expresiones enteras.

Versiones: 8K, Ampliado, Disco.

Propósito: Grabar un byte en una ubicación de memoria.

Observaciones: La expresión entera I es la dirección de la ubicación de memoria que ha de ser objeto de POKE. La expresión entera J es el dato. J debe estar dentro de los límites 0 a 255. En la versión 8K, I debe ser menor que 32768. En las versiones Ampliado y Disco, I ha de estar dentro de los límites 0 a 65536.

Con la versión 8K, los datos pueden ser POKE a ubicaciones de memoria por encima de 32768 proveyendo un número negativo para I. El valor de I se calcula restando 65536 de la dirección que se desea. Por ejemplo, para el POKE de datos a la ubicación 45000, I=45000-65536 ó −20536.

La función complementaria de POKE es PEEK. El argumento de PEEK es una dirección desde la cual ha de leerse un byte. (Ver Sección 3.27).

POKE y PEEK son útiles para un eficiente almacenamiento de datos, para subrutinas de carga en lenguaje Assembly y para transferir argumentos y resultados a y desde subrutinas en lenguaje Assembly.

Ejemplo: 10 POKE &H5AOO,&HFF

2.49 **PRINT**

Formato: PRINT ⟨lista de expresiones⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Salida de datos por la terminal.

Observaciones: Si se omitiera ⟨lista de expresiones⟩, se imprime una línea en blanco. Si se incluyera ⟨lista de expresiones⟩, se imprimen en la terminal los valores de las expresiones. Las expresiones de la lista pueden ser numéricas y/o de serie (las series deben encerrarse entre comillas).

Posiciones de Impresión

La puntuación utilizada para separar los ítems en la lista determina la posición de cada ítem impreso. El BASIC-80 divide la línea en zonas de impresión de 14 espacios cada una. Una coma, en la lista de expresiones, hace que el siguiente valor se imprima al comienzo de la siguiente zona. Un punto y coma imprime el siguiente valor inmediatamente después del último valor. La digitación de uno o más espacios entre expresiones tiene el mismo efecto que la digitación de un punto y coma.

Si una coma o un punto y coma terminara la lista de expresiones, la siguiente sentencia PRINT comienza a imprimir en la misma línea, en función del espaciado. Si la lista de expresiones terminara sin una coma o un punto y coma, al final de la línea se imprime un retorno de carro. Si la línea impresa fuese de longitud mayor que el ancho de la terminal, el BASIC-80 se dirige a la siguiente línea física y continúa la impresión.

Los números impresos son siempre seguidos por un espacio. Los números positivos son precedidos por un espacio. Los números negativos son precedidos por un signo menos. Los números de precisión simple que pueden representarse con 6 ó menos dígitos en el formato sin escala con menos precisión que con la que pueden representarse en el formato con escala, se imprimen usando el formato sin escala. Por ejemplo, 10∧(—6) tiene salida como .000001 y 10∧(—7) como 1E—7. Los números de precisión doble que pueden representarse con 16 ó menos dígitos en el formato sin escala con menos precisión que con la que pueden representarse en el formato con escala, se imprimen usando el formato sin escala. Por ejemplo, 1D—16 tiene salida como .0000000000000001 y 1D—17 como 1D—17.

En una sentencia PRINT puede usarse un signo de interrogación en lugar de la palabra PRINT.

Ejemplo 1:

```
10 X=5
20 PRINT X+5, X—5, X*(—5), X∧5
30 END
RUN
 10        0        —25        3125
Ok
```

En este ejemplo, las comas de la sentencia PRINT hacen que cada uno de los valores se imprima al comienzo de la siguiente zona de impresión.

Ejemplo 2:

```
LIST
10 INPUT X
20 PRINT X "AL CUADRADO ES" X∧2 "Y";
30 PRINT X "AL CUBO ES" X∧3
40 PRINT
50 GOTO 10
Ok
RUN
? 9
 9 AL CUADRADO ES 81 Y 9 AL CUBO ES 729
? 21
```

```
 21 AL CUADRADO ES 441 Y 21 AL CUBO ES 9261
?
```

En este ejemplo, el punto y coma al final de la línea 20 hace que ambas sentencias PRINT impriman en la misma línea, y la línea 40 imprime una línea en blanco antes de la siguiente indicación.

Ejemplo 3:

```
10 FOR X = 1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT X
Ok
RUN
 5  10  10  20  15  30  20  40  25  50
Ok
```

En este ejemplo, los puntos y comas de la sentencia PRINT hacen que cada uno de los valores sea impreso inmediatamente después del valor precedente. (No olvide, un número es siempre seguido por un espacio y los números positivos son precedidos por un espacio). En la línea 40 se usa un signo de interrogación en lugar de la palabra PRINT.

2.50 PRINT USING

Formato: PRINT USING ⟨expresión serie⟩; ⟨lista de expresiones⟩

Versiones: Ampliado, Disco.

Propósito: Imprimir series o números usando un formato especificado.

Observaciones y ejemplos: ⟨lista de expresiones⟩ consta de las expresiones de serie o numéricas que han de imprimirse, separadas mediante puntos y comas. ⟨expresión serie⟩ es un literal de serie (o variable) que consta de caracteres especiales de formato. Estos caracteres de formato (ver más adelante) determinan el campo y el formato de las series o números impresos.

Campos de Serie

Cuando se usa PRINT USING para imprimir series, para dar formato al campo puede utilizarse uno de tres caracteres de formato:

"!" Especifica que sólo ha de imprimirse el primer caracter de la serie dada.

"\n espacios\" Especifica que han de imprimirse 2 + n caracteres de la serie. Si las barras inversas se digitaran sin espacios, se imprimirán dos caracteres; con un espacio se imprimirán tres caracteres, y así sucesivamente. Si la serie fuese más larga que el campo, se ignoran los caracteres adicionales. Si el campo fuese más largo que la serie, la serie se justificará a la izquierda en el campo y rellenará con espacios a la derecha. Ejemplo:

```
10 A$="LOOK":B$="OUT"
30 PRINT USING "!";A$;B$
40 PRINT USING "\\";A$;B$
50 PRINT USING "\\";A$;B$;"!!"
RUN
LO
LOOKOUT
LOOK OUT !!
```

"&" Especifica un campo de serie de longitud variable. Cuando el campo se especifica con "&", la serie tiene una salida exactamente igual que la entrada. Ejemplo:

```
10 A$="LOOK":B$="OUT"
20 PRINT USING "/";A$;
30 PRINT USING "&";B$
RUN
LOUT
```

Campos Numéricos

Cuando PRINT USING se utiliza para imprimir números, para dar formato al campo numérico pueden emplearse los siguientes caracteres especiales:

≠ Para representar cada una de las posiciones de dígito se usa un signo numeral. Las posiciones de dígito se llenan siempre. Si el número a imprimir tuviera menos dígitos que las posiciones especificadas, el número se justificará a la derecha (precedido por espacios) del campo.

. Puede insertarse un punto decimal en cualquier posición del campo. Si la serie de formato especifica que al punto decimal ha de preceder un dígito, siempre se imprimirá el dígito (como 0 si fuese necesario). Los números se redondean como fuese necesario.

```
PRINT USING "≠≠.≠≠";.78
 0.78
PRINT USING "≠≠≠.≠≠";987.654
987,65
PRINT USING "≠≠.≠≠    ";10.2,5.3,66.789,.234
10.20   5.30   66.79   0.23
```

En el último ejemplo se insertaron tres espacios al final de la serie de formato para separar los valores impresos en la línea.

\+ Un signo más al comienzo o al final de la serie de formato imprimirá el signo del número (más o menos) delante o después del número.

— Un signo menos al final del campo de formato imprimirá números negativos con un signo menos a la derecha.

```
PRINT USING "+≠≠.≠≠    ";—68.95,2.4,55.6,—.9
—68.95   +2.40   +55.60   —0.90
PRINT USING "≠≠.≠≠—    ";—68.95,22.449,—7.01
68.95—   22.45   7.01—
```

** Un asterisco doble al comienzo de la serie de formato rellena con asteriscos los espacios a la izquierda del campo numérico. Los ** especifican asimismo posiciones para dos dígitos más.

```
PRINT USING "**≠.≠    ";12.39,—0.9,765.1
*12.4   *—0.9   765.1
```

$$ Un signo pesos doble imprime un signo pesos a la izquierda inmediata del número con formato. $$ especifica dos posiciones más de dígito, una de las cuales es el signo pesos. Con $$ no puede usarse el formato exponencial. No pueden usarse números negativos salvo que el signo menos esté a la derecha.

```
PRINT USING "$$≠≠≠.≠≠";456.78
$456.78
```

**$ **$ al comienzo de una serie de formato combina los efectos de los dos símbolos anteriores. Los espacios a la izquierda se rellenarán con asterisco y se imprimirá un signo pesos delante del número. **$ especifica tres posiciones de dígito más, una de las cuales es el signo pesos.

```
PRINT USING "**$≠≠.≠≠";2.34
***$2.34
```

, Una coma a la izquierda del punto decimal en una serie de formato imprime una coma a la izquierda de cada tercer dígito a la izquierda del punto decimal. Una coma al final de la serie de formato se imprime como parte de la serie. Una coma especifica otra posición de dígito. La coma carece de efecto si se emplea con el formato exponencial.

```
PRINT USING "≠≠≠≠,.≠≠";1234.5
1,234.50
PRINT USING "≠≠≠≠.≠≠,";1234.5
1234.50,
```

^^^^ Pueden colocarse cuatro flechas hacia arriba después de los caracteres de posición de dígito para especificar formato exponencial. Las cuatro flechas dejan espacio para imprimir E+xx. Puede especificarse cualquier posición de punto decimal. Los dígitos significativos se justifican a la izquierda y se ajusta el exponente. Salvo que se especificara un + inicial o + o — final, se usará una posición de dígito a la izquierda del punto decimal para imprimir un espacio o un signo menos.

```
PRINT USING "≠≠.≠≠^^^^";234.56
 2.35E+02
```

PRINT USING ".≠≠≠≠^^^^—";888888
.8889E+06
PRINT USING "+.≠≠^^^^";123
+.12E+03

— Un subrayado en la serie de formato da salida al siguiente caracter como un caracter literal.
PRINT USING "—!≠≠.≠≠—!";12.34
!12.34!
El caracter de literal en sí mismo puede ser un subrayado colocando "__ __" en la serie de formato.

% Si el número a imprimir fuese más largo que el campo numérico especificado, delante del número se imprime un signo porciento. Si el redondeo hiciese que el número exceda el campo, se imprimirá un signo porciento delante del número redondeado.
PRINT USING "≠≠.≠≠";111.22
%111.22
PRINT USING ".≠≠";.999
%1.00
Si el número de dígitos especificado excediera de 24, resultará un error "Llamado inválido de función".

2.51 PRINT≠ Y PRINT≠ USING

Formato: PRINT≠ ⟨número archivo⟩, [USING ⟨expr serie⟩;] ⟨lista de expresiones⟩

Versión: Disco.

Propósito: Grabar datos en un archivo secuencial en disco.

Observaciones: ⟨número archivo⟩ es el número empleado al abrirse el archivo para salida. ⟨expr serie⟩ consta de caracteres de formato, tal como se describe en la Sección 2.50, PRINT USING. Las expresiones de ⟨lista de expresiones⟩ son las expresiones numéricas y/o de serie que se grabarán en el archivo. PRINT≠ no comprime datos en el disco. En el disco se graba una imagen de los datos, tal como se representaría en la terminal con una sentencia PRINT. Por este motivo, se tomarán recaudos para delimitar los datos en el disco, para que ingresen correctamente desde él.

En la lista de expresiones, las expresiones numéricas se delimitarán con puntos y comas. Por ejemplo:

PRINT≠1,A;B;C;X;Y;Z

(Si como delimitadores se emplearan comas, los blancos adicionales que se inserten entre los campos de impresión también se grabarán en disco).

Las expresiones de serie deben separarse con puntos y comas en la lista. Para dar un formato correcto a las expresiones de serie en el disco, en la lista de expresiones use delimitadores explícitos.

Por ejemplo, sea A$="CAMARA" y B$="93604-1". La sentencia

PRINT≠1,A$;B$

Grabaría en disco CAMARA93604-1. Puesto que no hay delimitadores, esto no podría ingresarse como dos series separadas. Para salvar el problema, inserte delimitadores explícitos en la sentencia PRINT≠ de la siguiente manera:

PRINT≠1,A$;";";B$

La imagen grabada en disco es

CAMARA,93604-1

Si las series en sí mismas contuvieran comas, puntos y comas, blancos significativos a la izquierda, retornos de carro o alimentaciones de línea, grábelas en disco encerradas entre comillas explícitas, CHR$(34).

Por ejemplo, sea A$="CAMARA, AUTOMATICA" y B$=" 93694-1". La sentencia

PRINT≠1,A$;B$

grabaría en disco la siguiente imagen:

CAMARA, AUTOMATICA 93604-1

y la sentencia

INPUT≠1,A$,B$

ingresaría "CAMARA" en A$ y "AUTOMATICA 93604-1" en B$. Para separar adecuadamente estas series en disco, grabe comillas en la imagen en disco usando CHR$(34). La sentencia

PRINT≠1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
graba la siguiente imagen en disco:
"CAMARA, AUTOMATICA"" 93604-1"
y la sentencia
INPUT≠1,A$,B$
ingresaría "CAMARA, AUTOMATICA" en A$ y " 93604-1" en B$.
La sentencia PRINT≠ puede también usarse con la opción USING para controlar el formato del archivo en disco. Por ejemplo:
PRINT≠1,USING"$$≠≠≠.≠≠,";J;K;L
Ver en el Anexo B más ejemplos del uso de PRINT≠.
Ver asimismo WRITE≠, Sección 2.68.

2.52 PUT

Formato: PUT [≠] ⟨número archivo⟩ [,⟨número registro⟩]
Versión: Disco.
Propósito: Grabar un registro desde un buffer al azar en un archivo en disco al azar.
Observaciones: ⟨número archivo⟩ es el número con el cual se abrió el archivo. Si se omitiera ⟨número registro⟩, el registro tendrá el siguiente número de registro disponible (después del último PUT). El número mayor posible de registro es 32767.
Ejemplo: Ver Anexo B.

2.53 **RANDOMIZE**

Formato: RANDOMIZE [⟨expresión⟩]
Versiones: Ampliado, Disco.
Propósito: Dar nuevo origen al generador de números al azar.
Observaciones: Si se omitiera ⟨expresión⟩, el BASIC-80 suspende la ejecución del programa y solicita un valor imprimiendo

    Random Number Seed (0-65529)?

antes de ejecutar RANDOMIZE.
Si no diera nuevo origen al generador de números al azar, la función RND devuelve la misma secuencia de números al azar toda vez que se corre el programa (RUN). Para modificar la secuencia de números al azar toda vez que se corre el programa, coloque una sentencia RANDOMIZE al comienzo del programa y modifique el argumento con cada RUN.
Ejemplo:

```
10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RND;
40 NEXT I
RUN
Random Number Seed (0-65529)? 3 (el usuario digita 3)
 .88598   .484668   .586328   .119426   .709225
Ok
RUN
Random Number Seed (0-65529)? 4 (el usuario digita 4 para la nueva secuencia)
 .803506   .162462   .929364   .292443   .322921
Ok
RUN
Random Number Seed (0-65529)? 3 (la misma secuencia que el primer RUN)
 .88598   .484668   .586328   .119426   .709225
Ok
```

NOTA: Con el Compilador BASIC, la indicación que da RANDOMIZE es:
    Random Number Seed (—32768 to 32767)?

0 a → 65529

2.54 **READ**

Formato: READ ⟨lista de variables⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Leer valores de una sentencia DATA y asignarlos a variables. (Ver DATA, Sección 2.10).

Observaciones: Una sentencia READ debe usarse siempre en conjunción con una sentencia DATA. Las sentencias READ asignan variables a los valores de la sentencia DATA según una base uno a uno. Las variables de la sentencia READ pueden ser numéricas o de serie, y los valores leídos deben concordar con los tipos de variable especificados. Si no concordaran, resultará un "Error de Sintaxis".

Una sentencia READ puede acceder una o más sentencias DATA (se accederán en orden) o varias sentencias READ pueden acceder la misma sentencia DATA. Si el número de variables de ⟨lista de variables⟩ excediera el número de elementos de la(s) sentencia(s) DATA, se imprime un mensaje OUT OF DATA. Si el número de variables especificadas fuese menor que el número de elementos de la(s) sentencia(s) DATA, las subsiguientes sentencias READ comenzarán leyendo datos en el primer elemento sin leer. Los datos adicionales se ignoran si no hubiera subsiguientes sentencias READ.

Para volver a leer sentencias DATA desde el comienzo, use la sentencia RESTORE. (Ver RESTORE, Sección 2.57).

Ejemplo 1:

```
.
.
.
80 FOR I=1 TO 10
90 READ A(I)
100 NEXT I
110 DATA 3.08,5.19,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.37
.
.
.
```

Este segmento de programa lee (READ) los valores de las sentencias DATA en la matriz A. Después de la ejecución, el valor de A(1) será 3.08, y así sucesivamente.

Ejemplo 2:

```
LIST
10 PRINT "CITY", "STATE", "ZIP"
20 READ C$,S$,Z
30 DATA "DENVER,", COLORADO, 80211
40 PRINT C$,S$,Z
Ok
RUN
CITY            STATE           ZIP
DENVER,         COLORADO        80211
Ok
```

Este programa lee datos de serie y numéricos desde la sentencia DATA de la línea 30.

2.55 **REM**

Formato: REM ⟨comentarios⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Permitir la inserción de comentarios u observaciones en un programa.

Observaciones: Las sentencias REM no se ejecutan, pero tienen una salida igual a su ingreso al listarse el programa.

Puede bifurcarse a las sentencias REM (desde una sentencia GOTO o GOSUB) y la ejecución continuará en la primera sentencia ejecutable después de la sentencia REM.

En las versiones Ampliado y Disco, pueden agregarse comentarios al final de una línea precediéndola con un apóstrofo en lugar de :REM.

Ejemplo:

```
.
.
```

```
.
120 REM CALCULAR VELOCIDAD PROMEDIO
130 FOR I=1 TO 20
140 SUM=SUM + V(I)
.
.
.
```

o, con las versiones Ampliado y Disco:

```
.
.
.
120 FOR I=1 TO 20  'CALCULAR VELOCIDAD PROMEDIO
130 SUM=SUM+V(I)
140 NEXT I
.
.
.
```

2.56 **RENUM**

Formato: RENUM [[⟨nuevo número⟩] [,[⟨número anterior⟩] [,⟨incremento⟩]]]

Versiones: Ampliado, Disco.

Propósito: Renumerar líneas de programa.

Observaciones: ⟨nuevo número⟩ es el primer número de línea a usar en la nueva secuencia. El valor por defecto es 10. ⟨número anterior⟩ es la línea del programa corriente en la que ha de comenzar la renumeración. El valor por defecto es la primera línea del programa. ⟨incremento⟩ es el incremento a usar en la nueva secuencia. El valor por defecto es 10.

RENUM modifica asimismo todas las referencias a números de líneas que siguen a las sentencias GOTO, GOSUB, THEN, ON . . . GOTO, ON . . . GOSUB y ERL para reflejar los nuevos números de línea. Si después de una de estas sentencias apareciera un número de línea no existente, se imprime el mensaje de error "Línea indefinida xxxxx en yyyyy". RENUM no modifica la referencia incorrecta al número de línea (xxxxx), pero puede modificarse el número de línea yyyyy.

NOTA: No puede usarse RENUM para modificar el orden de las líneas del programa (por ejemplo, RENUM 15,30 cuando el programa tiene tres líneas numeradas 10, 20 y 30 ó para crear números de líneas mayores que 65529. Resultará un error "Llamado de función inválido".

Ejemplos:

RENUM	Renumera todo el programa. El nuevo número de la primera línea será 10. Las líneas incrementarán en 10.
RENUM 300,,50	Renumera todo elprograma. El nuevo número de la primera línea será 300. Las líneas incrementarán en 50.
RENUM 1000,900,20	Renumera las líneas a partir de la 900 con el número de línea 100 y con un incremento de 20.

2.57 **RESTORE**

Formato: RESTORE ⟨número línea⟩

Versiones: 8K, Ampliado, Disco.

Propósito: Volver a leer sentencias DATA desde un punto especificado.

Observaciones: Luego de ejecutarse una sentencia RESTORE, la siguiente sentencia READ accede el primer ítem de la primer sentencia DATA del programa. Si se especificara ⟨número línea⟩, la siguiente sentencia READ accede el primer ítem de la sentencia DATA especificada.

Ejemplo:

```
10 READ A,B,C
20 RESTORE
```

```
30 READ D,E,F
40 DATA 57, 68, 79
.
.
.
```

2.58 **RESUME**

Formatos: RESUME
RESUME 0
RESUME NEXT
RESUME ⟨número línea⟩

Versiones: Ampliado, Disco.

Propósito: Continuar la ejecución del programa después de llevarse a cabo un procedimiento de recuperación de errores.

Observaciones: Puede usarse cualquiera de los cuatro formatos anteriores, en función del lugar en el que ha de reanudarse la ejecución:

RESUME	La ejecución se reanuda en la sentencia que provocó el error.
o	
RESUME 0	
RESUME NEXT	La ejecución se reanuda en la sentencia que sigue de inmediato a la que provocó el error.
RESUME ⟨número línea⟩	La ejecución se reanuda en ⟨número línea⟩.

Una sentencia RESUME que no esté en una rutina de desvío por error imprime el mensaje "RESUME sin error".

Ejemplo:

```
10 ON ERROR GOTO 900
.
.
.
900 IF (ERR=230)AND(ERL=90) THEN PRINT "PRUEBE
OTRA VEZ":RESUME 80
.
.
.
```

2.59 **RUN**

Formato 1: RUN [⟨número línea⟩]

Versiones: 8K, Ampiado, Disco.

Propósito: Ejecutar el programa corrientemente en memoria.

Observaciones: Si se especificara ⟨número línea⟩, la ejecución comienza en esa línea. En caso contrario, la ejecución comienza en el número más bajo de línea. El BASIC-80 vuelve siempre a nivel de comando después de ejecutarse RUN.

Ejemplo: RUN

Formato 2: RUN ⟨nombrearchivo⟩ ,R

Versión: Disco.

Propósito: Carga un archivo de disco en memoria y ejecutarlo.

Observaciones: ⟨nombrearchivo⟩ es el nombre usado al resguardarse (SAVE) el archivo. (Con CP/M e ISIS-II, se provee la extensión por defecto .BAS).

RUN cierra todos los archivos abiertos y elimina el contenido corriente de la memoria antes de cargar el programa designado. Sin embargo, con la opción "R", todos los archivos de datos siguen abiertos.

Ejemplo: RUN "NEWFIL" ,R
Ver asimismo Anexo B.

2.60 **SAVE**

Formato: SAVE ⟨nombrearchivo⟩ [,A | ,P]

Versión: Disco.

Propósito: Resguardar un programa en disco.

Observaciones: ⟨nombrearchivo⟩ es una serie entre comillas que observa los requerimientos de su sistema operativo en cuanto a nombres de archivos. (Con CP/M, se provee la extensión por defecto .BAS). Si ya existiera ⟨nombrearchivo⟩, el archivo se sobregrabará.

Emplee la opción *A* para resguardar el archivo en formato ASCII. En caso contrario, el BASIC resguardará el archivo en formato primario comprimido. El formato ASCII lleva más espacio en disco, pero ciertos accesos a disco requieren que los archivos estén en formato ASCII. Por ejemplo, el comando MERGE requiere un archivo con formato ASCII, y algunos comandos del sistema operativo tales como LIST pueden requerir un archivo con formato ASCII.

Emplee la opción P para proteger al archivo resguardándolo en un formato codificado en binario. Cuando un archivo protegido es posteriormente RUN (o LOAD), fallará todo intento de listarlo o editarlo.

Ejemplos: SAVE"COM2",A
SAVE"PROG",P
Ver asimismo Anexo B.

2.61 **STOP**

Formato: STOP.

Versiones: 8K, Ampliado, Disco.

Propósito: Terminar la ejecución del programa y volver a nivel de comando.

Observaciones: Las sentencias STOP pueden usarse en cualquier lugar de un programa para terminar su ejecución. Al detectarse una STOP, se imprime el siguiente mensaje:

Break in line nnnnn
(Interrupción en línea nnnnn)

Contrariamente a la sentencia END, la sentencia STOP no cierra los archivos.

El BASIC-80 vuelve siempre a nivel de comando después de ejecutarse STOP. La ejecución se reanuda ejecutando un comando CONT. (Ver Sección 2.8).

Ejemplo:

```
10 INPUT A,B,C
20 K=A 2*5.3:L=B^3/.26
30 STOP
40 M=C*K+100:PRINT M
RUN
? 1,2,3
BREAK IN 30
Ok
PRINT L
 30.7692
Ok
CONT
 115.9
Ok
```

2.62 SWAP

Formato: SWAP ⟨variable⟩, ⟨variable⟩

Versiones: Ampliado, Disco.

Propósito: Intercambiar los valores de dos variables.

Observaciones: Puede intercambiarse una variable de cualquier tipo (entera, precisión simple, precisión doble, serie), pero las dos variables han de ser del mismo tipo, o se producirá un error "Falta de concordancia en tipo".

Ejemplo:

```
LIST
10 A$=" UNO " : B$=" TODOS " : C$=" RA"
20 PRINT A$ C$ B$
30 SWAP A$, B$
40 PRINT A$ C$ B$
RUN
Ok
 UNO PARA TODOS
 TODOS PARA UNO
Ok
```

2.63 TRON/TROFF

Formato: TRON
TROFF

Versiones: Ampliado, Disco.

Propósito: Rastrear la ejecución de sentencias de programa.

Observaciones: Como ayuda en la corrección de errores, la sentencia TRON (ejecutada en modalidad directa o indirecta) activa una señal de rastreo que imprime cada uno de los números de las líneas del programa a medida que se ejecutan. Los números aparecen entre corchetes. La señal de rastreo se inactiva con la sentencia TROFF (o al ejecutarse un comando NEW).

Ejemplo:

```
TRON
Ok
LIST
10 K=10
20 FOR J=1 TO 2
30 L=K+10
40 PRINTJ;K;L
50 K=K+10
60 NEXT
70 END
Ok
RUN
[10] [20] [30] [40] 1  10  20
[50] [60] [30] [40] 2  20  30
[50] [60] [70]
Ok
TROFF
Ok
```

2.64 WAIT

Formato: WAIT ⟨número vía⟩,I[,J]
en donde I y J son expresiones enteras.

Versiones: 8K, Ampliado, Disco.

Propósito: Suspender la ejecución del programa mientras se supervisa el estado de una vía de entrada de máquina.

Observaciones: La sentencia WAIT suspende la ejecución hasta que una vía de entrada de máquina desarrolla un J y luego AND a I. Si el resultado fueseeídos en la vía son OR exclusivo con la expresión entera modelo de bits especificado. Los datos I cero, el BASIC-80 retorna y lee nuevamente los datos de la vía. Si el resultado fuera no cero, la e,ecución continúa con la siguiente sentencia. Si se omitiera J, se presupone ser cero.

Precaución: Con la sentencia WAIT es posible ingresar en un lazo infinito, en cuyo caso será necesario reiniciar manualmente la máquina.

Ejemplo: 100 WAIT 32,2

2.65 **WHILE . . . WEND**

Formato:

```
WHILE ⟨expresión⟩
.
.
[⟨sentencias del lazo⟩]
.
.
WEND
```

Versiones: Ampliado, Disco.

Propósito: Ejecutar una serie de sentencias de un lazo en tanto una condición dada sea verdadera.

Observaciones: Si ⟨expresión⟩ no fuera cero (esto es, verdadera), las sentencias del lazo se ejecutan hasta detectar la sentencia WEND. El BASIC vuelve luego a la sentencia WHILE y comprueba la ⟨expresión⟩. Si sigue siendo verdadera, el proceso se repite. Si no lo fuese, la ejecución se reanula en la sentencia que sigue a la WEND.

Los lazos WHILE/WEND pueden anidarse a cualquier nivel. Todo WEND concordará con el WHILE más reciente. Una sentencia WHILE sin concordancia provoca un error "WHILE sin WEND"; una sentencia WEND sin concordancia provoca un error "WEND sin WHILE".

Ejemplo:

```
90 'BUBBLE SORT ARRAY A$
100 FLIPS=1 'FORCE ONE PASS THRU LOOP
110 WHILE FLIPS
115         FLIPS=0
120         FOR I=1 TO J-1
130                 IF A$(I)>A$(++1) THEN
                            SWAP A$(I),A$(I+1):FLIPS=1
140         NEXT I
WEND
150 WEND
```

2.66 **WIDTH**

Formato: WIDTH [LPRINT] ⟨expresión entera⟩

Versiones: Ampliado, Disco.

Propósito: Fijar el número de caracteres del ancho de la línea impresa en la terminal o en la impresora de líneas.

Observaciones: Si se omitiera la opción LPRINT, el ancho de la línea se fija en la terminal. Si se incluye LPRINT, el ancho de la línea se fija en la impresora de líneas.

⟨expresión entera⟩ debe tener un valor entre los límites 15 a 255. El ancho por defecto es de 72 caracteres.

Si ⟨expresión entera⟩ fuese 255, el ancho de la línea es "infinito", esto es, el BASIC nunca inserta un retorno de carro. Sin embargo, la posición del cursor o de la cabeza de impresión, tal como la da la función POS o LPOS, retorna a cero después de la posición 255.

2.67 **WRITE**

Formato: WRITE [⟨lista de expresiones⟩]

Versión: Disco.

Propósito: Dar salida a datos en la terminal.

Observaciones: Si se omitiera ⟨lista de expresiones⟩, se da salida a una línea en blanco. Si se incluye ⟨lista de expresiones⟩, los valores de las expresiones tienen salida por terminal. Las expresiones de la lista pueden ser numéricas y/o de serie, y han de estar separadas con comas.

Al darse salida a los ítems impresos, cada uno de los ítems se separará del último con una coma. Las series impresas se delimitarán con comillas. Tras imprimirse el último ítem de la lista, el BASIC inserta un retorno de carro/alimentación de línea.

WRITE da salida a valores numéricos usando el mismo formato que la sentencia PRINT, Sección 2.49.

Ejemplo:

```
10 A=80;B=90:C$=ESO ES TODO
20 WRITE A,B,C$
RUN
 80,90, "ESO ES TODO"
Ok
```

2.68 **WRITE≠**

Formato: WRITE≠ ⟨número archivo⟩, ⟨lista de expresiones⟩

Versión: Disco.

Propósito: Grabar datos en un archivo secuencial.

Observaciones: ⟨número archivo⟩ es el número con el que el archivo fue abierto en modalidad "0". Las expresiones de la lista son de serie o numéricas y han de estar separadas con comas.

La diferencia entre WRITE≠ y PRINT≠ radica en que WRITE≠ inserta comas entre los ítems al grabarse en disco y delimita las series con comillas. Por lo tanto, no es necesario que el usuario coloque delimitadores explícitos en la lista. Después de grabarse en disco el último ítem de la lista se inserta un retorno de carro/alimentación de línea.

Ejemplo: Sea A$="CAMARA" y B$="93604-1". La sentencia

```
WRITE≠1,A$,B$
```

graba la siguiente imagen en disco:

```
"CAMARA","93604-1"
```

Una subsiguiente sentencia INPUT≠, tal como:

```
INPUT≠1,A$,B$
```

ingresaría "CAMARA" en A$ y "93604-1" en B$.

CAPITULO 3

FUNCIONES DEL BASIC-80

Presentamos en este capítulo las funciones intrínsecas que provee el BASIC-80. Las funciones pueden llamarse desde cualquier programa sin definición adicional.

Los argumentos de las funciones se encierran siempre entre paréntesis. En los formatos dados a las funciones de este capítulo, los argumentos han sido abreviados de la siguiente manera:

X e Y	Representan cualesquiera expresiones numéricas.
I y J	Representan expresiones enteras.
X$ e Y$	Representan expresiones de serie.

Si cuando se requiriera un entero se suministrara un valor en punto flotante, el BASIC-80 redondeará la porción fraccionaria y usará el entero resultante.

NOTA: Con los intérpretes BASIC-80 y BASIC-86, las funciones sólo devuelven resultados enteros y de precisión simple. Las funciones de precisión doble son apoyadas solamente por el compilador BASIC.

3.1 ABS

Formato: ABS(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el valor absoluto de la expresión X.

Ejemplo:
```
PRINT ABS(7*(—5))
 35
Ok
```

3.2 ASC

Formato: ASC(X$)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve un valor numérico que es el código ASCII del primer caracter de la serie X$. Si X$ fuese nulo, se devuelve un error "Llamado de función inválido".

Ejemplo:
```
10 X$ = "TEST"
20 PRINT ASC(X$)
RUN
 84
Ok
```
Ver en la función CHR$ la conversión ASCII a serie.

3.3 ATN

Formato: ATN(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el arcotangente de X en radianes. El resultado se da entre los límites —pi/2 a pi/2. La expresión X puede ser cualquier tipo numérico, pero la evaluación de ATN se hace siempre con precisión simple.

Ejemplo:
```
10 INPUT X
20 PRINT ATN(X)
RUN
? 3
 1.24905
Ok
```

3.4 **CDBL**

Formato: CDBL(X)

Versiones: Ampliado, Disco.

Acción: Convierte X a un número de precisión doble.

Ejemplo:
```
10A = 454.67
20 PRINT A;CDBL(A)
RUN
 454.67   454.6700134277344
Ok
```

3.5 **CHR$**

Formato: CHR$(I)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve una serie cuyo elemento uno tiene I en código ASCII. (Los Códigos ASCII se listan en el Anexo L.) CHR$ se usa comúnmente para transmitir un caracter especial a la terminal. Por ejemplo, podría transmitirse el caracter BEL (CHR$(7)) como prefacio a un mensaje de error, o bien una alimentación de formulario (CHR$(12)) para borrar una pantalla CRT y volver el cursor a la posición inicial.

Ejemplo:
```
PRINT CHR$(66)
B
Ok
```
Ver en la función ASC la conversión ASCII a numérico.

3.6 **CINT**

Formato: CINT(X)

Versiones: Ampliado, Disco.

Acción: Convierte X a entero redondeando la porción fraccionaria. Si X no estuviera dentro de los límites −32768 a 32767, se produce un error de "Overflow".

Ejemplo:
```
PRINT CINT(45.67)
 46
Ok
```
Ver en las funciones CDBL y CSNG la conversión de números al tipo de datos de precisión doble y precisión simple. Ver asimismo las funciones FIX e INT, que devuelven enteros.

3.7 **COS**

Formato: COS(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el coseno de X en radianes. El cálculo de COS(X) se hace con precisión simple.

Ejemplo:
```
10 X = 2*COS(.4)
20 PRINT X
```

```
RUN
 1.84212
Ok
```

3.8 **CSNG**

Formato: CSNG(X)

Versiones: Ampliado, Disco.

Acción: Convierte X a un número de precisión simple.

Ejemplo:
```
10 A≠ = 975.3421≠
20 PRINT A≠; CSNG(A≠)
RUN
 975.3421   975.342
Ok
```

Ver en las funciones CINT y CDBL la conversión de números a tipos de datos entero y precisión doble.

3.9 **CVI, CVS, CVD**

Formato: CVI (⟨serie 2 bytes⟩)
CVS (⟨serie 4 bytes⟩)
CVD (⟨serie 8 bytes⟩)

Versión: Disco.

Acción: Convierte valores de serie a valores numéricos. Los valores numéricos que se leen desde un archivo en disco al azar deben convertirse nuevamente de series a números. CVI convierte una serie de 2 bytes a un entero. CVS convierte una serie de 4 bytes a un número de precisión simple. CVD convierte una serie de 8 bytes a un número de precisión doble.

Ejemplo:
```
.
.
.
70 FIELD ≠1,4 AS N$, 12 AS B$, ...
80 GET ≠1
90 Y=CVS(N$)
.
.
.
```

Ver asimismo MKI$, MKS$, MKD$, Sección 3.25 y Anexo B.

3.10 **EOF**

Formato: EOF (⟨número archivo⟩)

Versión: Disco.

Acción: Devuelve —1 (verdadero) si se ha alcanzado el fin de un archivo secuencial. Use EOF para probar fin de archivo al hacer INPUT, para evitar errores "Entrada más allá del fin".

Ejemplo:
```
10 Open "I",1,"DATA"
20 C=0
30 IF EOF(1) THEN 100
40 INPUT ≠1,M(C)
50 C=C+1:GOTO 30
.
.
.
```

3.11 **EXP**

Formato: EXP(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve e a la potencia X. X debe ser <=87.3365. Si EXP excediera de capacidad, se representa el mensaje de error "Overflow"; como resultado se provee infinitud de máquina con el signo apropiado y la ejecución continua.

Ejemplo:
```
10 X = 5
20 PRINT EXP (X-1)
RUN
 54.5982
Ok
```

3.12 **FIX**

Formato: FIX(X)

Versiones: Ampliado, Disco.

Acción: Devuelve la parte entera truncada de X. FIX(X) es el equivalente de SGN(X)*INT(ABS(X)). La principal diferencia entre FIX e INT radica en que FIX no devuelve el siguiente número más bajo para X negativo.

Ejemplos:
```
PRINT FIX(58.75)
 58
Ok
PRINT FIX(-58.75)
-58
Ok
```

3.13 **FRE**

Formato: FRE(0)
FRE(X$)

Versiones: 8K, Ampliado, Disco.

Acción: Lor argumentos de FRE son ficticios. FRE devuelve el número de bytes de memoria que no está usando el BASIC-80.

FRE(" ") fuerza una recolección de información no válida ("basura") antes de devolver el número de bytes libres. SEA PACIENTE: la recolección de basura puede llevar de 1 minuto a 1,5 minutos. El BASIC no iniciará la recolección de la basura hasta que no se haya consumido toda la memoria libre. Por lo tanto, el uso periódico de FRE(" ") resultará en demoras más cortas por recolección de basura.

Ejemplo:
```
PRINT FRE(0)
 14542
Ok
```

3.14 **HEX$**

Formato: HEX$(X)

Versiones: Ampliado, Disco.

Acción: Devuelve una serie que representa el valor hexadecimal del argumento decimal. X se redondea a entero antes de evaluarse HEX$(X).

Ejemplo:
```
10 INPUT X
20 A$ = HEX$(X)
30 PRINT X "DECIMAL ES" A$ "HEXADECIMAL"
```

```
RUN
? 32
 32 DECIMAL ES 20 HEXADECIMAL
Ok
```

Ver en la función **OCT$** la conversión octal.

3.15 INP

Formato: INP(I)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el byte leído en la vía I. I debe estar entre los límites 0 a 255. INP es la función complementaria de la sentencia OUT (Sección 2.47.)

Ejemplo: 100 A=INP(255)

3.16 INPUT$

Formato: INPUT$(X[,[#]Y])

Versión: Disco.

Acción: Devuelve una serie de X caracteres, leídos desde la terminal o desde el archivo número Y. Si para entrada se usara la terminal, los caracteres no tendrán eco y se pasan todos los caracteres de control, excepto Control—C, que se usa para interrumpir la ejecución de la función INPUT$.

Ejemplo 1:
```
5 'LISTAR EL CONTENIDO DE UN ARCHIVO SECUENCIAL EN HEXADECIMAL
10 OPEN"I",1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Ejemplo 2:
```
.
.
.
100 PRINT "DIGITAR P PARA CONTINUAR O S PARA PARAR"
110 X$=INPUT$(1)
120 IF X$="P" THEN 500
130 IF X$="S" THEN 700 ELSE 100
.
.
.
```

3.17 INSTR

Formato: INSTR([I,]X$,Y$)

Versiones: Ampliado, Disco.

Acción: Busca la primera ocurrencia de la serie Y$ en X$ y devuelve la posición en la que se detecta la concordancia. El desplazamiento opcional I fija la posición para iniciar la búsqueda. I ha de estar entre los límites 0 y 255. Si I>LEN((X$) o si X$ fuese nulo o si no pudiera hallarse Y$, INSTR devuelve 0. Si Y$ fuese nulo, INSTR devuelve I ó 1. X$ e Y$ pueden ser variables de serie, expresiones de serie o literales de serie.

Ejemplo:
```
10 X$ = "ABCDEB"
20 Y$ = "B"
30 PRINT INSTR(X$,Y$);INSTR(4,X$,Y$)
```

```
RUN
 2 6
Ok
```

3.18 INT

Formato: INT(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el entero más grande =X.

Ejemplos:
```
PRINT INT(99.89)
 99
Ok
PRINT INT(—12.11)
—13
Ok
```
Ver las funciones FIX y CINT, que también devuelven valores enteros.

3.19 LEFT$

Formato: LEFT$(X$,I)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve una serie de los I caracteres de extrema izquierda de X$. I ha de estar en los límites 0 a 255. Si I fuera mayor que LEN(X$), se devolverá toda la serie (X$). Si I=0, se devuelve la serie nula (longitud cero).

Ejemplo:
```
10 A$ = "BASIC-80"
20 B$ = LEFT$(A$,5)
30 PRINT B$
BASIC
Ok
```
Ver asimismo las funciones MID$ y RIGHT$.

3.20 LEN

Formato: LEN(X$)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el número de caracteres de X$. Se cuentan los caracteres de no impresión y los blancos.

Ejemplo:
```
10 X$ = "PORTLAND, OREGON"
20 PRINT LEN(X$)
 16
Ok
```

3.21 LOC

Formato: LOC (⟨número archivo⟩)

Versión: Disco.

Acción: Con archivos en disco al azar, LOC devuelve el siguiente número de registro a usar si se ejecuta una GET o una PUT (sin número de registro). Con archivos secuenciales, LOC devuelve el número de sectores (bloques de 128 bytes) leídos desde o grabados en el archivo desde que fue abierto.

Ejemplo:
```
200 IF LOC(1)>50 THEN STOP
```

3.22 **LOG**

Formato: LOG(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el logaritmo natural de X. X debe ser mayor que 0.

Ejemplo:
```
PRINT LOG(45/7)
1.86075
Ok
```

3.23 **LPOS**

Formato: LPOS(X)

Versiones: Ampliado, Disco.

Acción: Devuelve la posición corriente de la cabeza de impresión dentro del buffer de la impresora de líneas. No da necesariamente la posición física de la cabeza de impresión. X es un argumento ficticio.

Ejemplo:
```
100 IF LPOS(X)>60 THEN LPRINT CHR$(13)
```

3.24 **MID$**

Formato: MID$(X$,I[,J])

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve una serie de J caracteres de longitud desde X$, comenzando con el lº carácter. I y J deben estar dentro de los límites 0 a 225. Si se omitiera J o si hubiera menos caracteres que J a la derecha del lº caracter, se devuelven todos los caracteres de la extrema derecha comenzando con el lº caracter. Si I LEN(X$), MID$ devuelve una serie nula.

Ejemplo:
```
LIST
10 A$="GOOD"
20 B$="MORNING EVENING AFTERNOON"
30 PRINT A$;MID$(B$,9,7)
Ok
RUN
GOOD EVENING
Ok
```
Ver asimismo las funciones LEFT$ y RIGHT$.

3.25 **MKI$, MKS$, MKD$**

Formato:
```
MKI$ (⟨expresión entera⟩)
MKS$ (⟨expresión de precisión simple⟩)
MKD$ (⟨expresión de precisión doble⟩)
```

Versión: Disco.

Acción: Convierte valores numéricos a valores de serie. Cualquier valor numérico que se coloque en un buffer de archivo al azar con una sentencia LSET o RSET debe convertirse a una serie. MKI$ convierte un entero a una serie de 2 bytes. MKS$ convierte un número de precisión simple a una serie de 4 bytes. MKD$ convierte un número de precisión doble a una serie de 8 bytes.

Ejemplo:
```
90 AMT=(K+T)
100 FIELD #1, 8 AS D$, 20 AS N$
110 LSET D$ = MKS$(AMT)
120 LSET N$ = A$
130 PUT #1
```

Ver asimismo CVI, CVS, CVD, Sección 3.9 y Anexo B.

3.26 **OCT$**

Formato: OCT$(X)

Versiones: Ampliado, Disco.

Acción: Devuelve una serie que representa el valor octal del argumento decimal. X se redondea a entero antes de evaluarse OCT$(X).

Ejemplo:
```
PRINT OCT$(24)
 30
Ok
```
Ver la función HEX$ para conversión hexadecimal.

3.27 **PEEK**

Formato: PEEK(I)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el byte (entero decimal en los límites 0 a 255) leído en la ubicación de memoria I. Con la versión 8K del BASIC-80, I debe ser menor que 32768. Para PEEK una ubicación de memoria por encima de 32768, réstese 65536 de la dirección deseada. Con el BASIC-80 Ampliado y Disco, I ha de estar dentro de los límites 0 a 65536. PEEK es la función complementaria de la sentencia POKE, Sección 2.48.

Ejemplo: A=PEEK(&H5A00)

3.28 **POS**

Formato: POS(I)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve la posición corriente del cursor. La posición de extrema izquierda es 1. X es un argumento ficticio.

Ejemplo: IF POS(X)>60 THEN PRINT CHR$(13)

Ver asimismo la función LPOS.

3.29 **RIGHT$**

Formato: RIGHT$(X$,I)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve los I caracteres de extrema derecha de la serie X$. Si I=LEN(X$), devuelve X$. Si I=0, se devuelve la serie nula (longitud cero).

Ejemplo:
```
10 A$="DISK BASIC-80"
20 PRINT RIGHT$(A$,8)
RUN
BASIC-80
Ok
```
Ver asimismo las funciones MID$ y LEFT$.

3.30 **RND**

Formato: RND[(X)]

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve un número al azar entre 0 y 1. Cada vez que el programa es ejecutado (RUN) se genera la misma secuencia de números al azar salvo que se dé nuevo origen al generador de números al azar (ver RANDOMIZE, Sección 2.53). Sin embargo, X<0 reinicia siempre la misma secuencia para cualquier X dada.

Si X>0 ó de omitirse X genera el siguiente número al azar de la secuencia. X=0 repite el último número generado.

Ejemplo:
```
10 FOR I=1 TO 5
20 PRINT INT(RND*100);
30 NEXT
RUN
 24   30   31   51   5
Ok
```

3.31 **SGN**

Formato: SGN(X)

Versiones: 8K, Ampliado, Disco.

Acción: Si X>0, SGN(X) devuelve 1.
Si X=0, SGN(X) devuelve 0.
Si X<0, SGN(X) devuelve —1.

Ejemplo: ON SGN(X)+2 GOTO 100,200,300 bifurca a 100 si X es negativo, a 200 si X es 0 y a 300 si X es positivo.

3.32 **SIN**

Formato: SIN(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el seno de X en radianes. SIN(X) se calcula con precisión simple.
COS(X)=SIN(X+3.14159/2).

Ejemplo:
```
PRINT SIN(1.5)
 .997495
Ok
```

3.33 **SPACE$**

Formato: SPACE$(X)

Versiones: Ampliado, Disco.

Acción: Devuelve una serie de espacios de longitud X. La expresión X se redondea a entero y debe estar entre los límites 0 a 255.

Ejemplo:
```
10 FOR I = 1 TO 5
20 X$ = SPACE$(I)
30 PRINT X$;I
40 NEXT I
RUN
  1
   2
    3
     4
      5
Ok
```

Ver asimismo la función SPC.

3.34 **SPC**

Formato: SPC(I)

Versiones: 8K, Ampliado, Disco.

Acción: Imprime I blancos en la terminal. SPC sólo puede usarse con sentencias PRINT y LPRINT. I ha de estar dentro de los límites 0 a 255.

Ejemplo:
```
PRINT "MAS" SPC(15) "ALLA"
MAS               ALLA
Ok
```
Ver asimismo la función SPACE$.

3.35 **SQR**

Formato: SQR(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve la raíz cuadrada de X. X debe ser >=0.

Ejemplo:
```
10 FOR X = 10 TO 25 STEP 5
20 PRINT X, SQR(X)
30 NEXT
RUN
 10          3.16228
 15          3.87298
 20          4.47214
 25          5
Ok
```

3.36 **STR$**

Formato: STR$(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve una representación de serie del valor de X.

Ejemplo:
```
5 REM ARITMETICA PARA NIÑOS
10 INPUT "DIGITE UN NUMERO";N
20 ON LEN(STR$(N)) GOSUB 30,100,200,300,400,500
.
.
.
```
Ver asimismo la función VAL.

3.37 **STRING$**

Formatos: STRING$(I,J)
STRING$(I,X$)

Versiones: Ampliado, Disco.

Acción: Devuelve una serie de longitud I cuyos caracteres tienen todos el código ASCII J o el primer caracter de X$.

Ejemplo:
```
10 X$ = STRING$(10,45)
20 PRINT X$ "INFORME MENSUAL" X$
RUN
----------INFORME MENSUAL----------
Ok
```

3.38 **TAB**

Formato: TAB(I)

Versiones: 8K, Ampliado, Disco.

Acción: Espacia a la posición I en la terminal. Si la posición de impresión corriente estuviera ya más allá de I, TAB se dirige a esa posición en la línea siguiente. Espacio 1 es la posición de extrema izquierda, y la posición de la extrema derecha es el ancho menos uno. I debe estar dentro de los límites 1 a 255. TAB sólo puede usarse en sentencias PRINT y LPRINT.

Ejemplo:
```
10 PRINT "NOMBRE" TAB(25) "IMPORTE" : PRINT
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "G. T. JONES", "$25.00"
RUN
NOMBRE                    IMPORTE
G. T. JONES               $25.00
Ok
```

3.39 **TAN**

Formato: TAN(X)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve la tangente de X en radianes. TAN(X) se calcula con precisión simple. Si TAN excediera de capacidad, se representa el mensaje de error "Overflow" y como resultado se provee infinitud de máquina con el signo apropiado, continuando la ejecución.

Ejemplo:
```
10 Y = Q*TAN(X)/2
```

3.40 **USR**

Formato: USR [⟨dígito⟩](X)

Versiones: 8K, Ampliado, Disco.

Acción: Llama a la subrutina en lenguaje Assembly del usuario con el argumento X. Se permite ⟨dígito⟩ solamente en las versiones Ampliado y Disco. Dígito está dentro de los límites 0 a 9 y corresponde al dígito provisto con la sentencia DEF USR para esa rutina. Si se omitiera ⟨dígito⟩, se presupone USR0. (Ver Anexo C.)

Ejemplo:
```
40 B = T*SIN(Y)
50 C = USR(B/2)
60 D = USR(B/3)
```

3.41 **VAL**

Formato: VAL(X$)

Versiones: 8K, Ampliado, Disco.

Acción: Devuelve el valor numérico de la serie X$. Si el primer caracter de X$ no fuese +, —, & o un dígito, VAL(X$)=0.

Ejemplo:
```
10 READ NAME$,CITY$,STATE$,ZIP$
20 IF VAL(ZIP$)<90000 OR VAL(ZIP$)>96699 THEN
PRINT NAME$ TAB(25) "FUERA DEL ESTADO"
30 IF VAL(ZIP$)>=90801 AND VAL(ZIP$)<=90815 THEN
PRINT NAME$ TAB(25) "LONG BEACH"
.
.
.
```

Ver conversión de numérico a serie en la función STR$.

3.42 **VARPTR**

Formato 1: VARPTR (⟨nombre variable⟩)

Versiones: Ampliado, Disco.

Formato 2: VARPTR (#⟨número archivo⟩)

Versión: Disco.

Acción: Formato 1: Devuelve la dirección del primer byte de datos identificados con ⟨nombre variable⟩. Debe asignarse un valor a ⟨nombre variable⟩ antes de la ejecución de VARPTR. De no ser así resulta un error "Llamado de función inválido". Puede usarse cualquier nombre de tipo de variable (numérica, de serie, de matriz) y la dirección devuelta será un entero dentro de los límites 32767 a —32768. Si se devolviera un número negativo, súmelo a 65536 para obtener la dirección real.

Por lo común VARPTR se usa para obtener la dirección de una variable o matriz para que pueda ser transferida a una subrutina en lenguaje Assembly. Usualmente se especifica un llamado de función del formato VARPTR(A(0)) al transferir una matriz, a fin de devolver el elemento de dirección más baja de la matriz.

NOTA: Se asignarán todas las variables simples antes de llamar a VARPTR para una matriz, puesto que las direcciones de las matrices cambian toda vez que se asigna una variable simple nueva.

Formato 2: Devuelve la dirección inicial del buffer de E/S en disco asignada a ⟨número archivo⟩.

En el BASIC Autónomo en Disco, VARPTR (#⟨número archivo⟩) devuelve el primer byte del bloque de archivo.

Ejemplo: 100 X = USR(VARPTR(Y))

ANEXO A

NUEVOS DISPOSITIVOS DEL BASIC-80, EDICION 5.0

La ejecución de los programas BASIC escritos bajo el BASIC Microsoft, edición 4.51 y anteriores, puede verse afectada por algunos de los nuevos dispositivos de la edición 5.0. Antes de intentar ejecutar tales programas, corrobore cuanto sigue:

1. Nuevas palabras reservadas: CALL, CHAIN, COMMON, WHILE, WEND, WRITE, OPTION BASE, RANDOMIZE.
2. La conversión de valores en punto flotante a enteros resulta en un redondeo y no en truncamiento. Esto afecta no sólo las sentencias de asignación (por ejemplo, I%=2.5 resulta en I%=3), sino también las evaluaciones de funciones y sentencias [por ejemplo, TAB(4.5) se dirige a la quinta posición, A(1.5) resulta en A(2) y X=11.5 MOD 4 resulta en 0 para X].
3. Se omite el cuerpo de un lazo FOR . . . NEXT si el valor inicial del lazo por el signo del paso excede el valor final por el signo del paso. (Ver Sección 2.22.)
4. La división por cero y el exceso de capacidad ("overflow") no producen ya errores fatales. (Ver Sección 1.8.1.2.)
5. Se ha modificado la función RND de modo que RND sin argumento es lo mismo que RND con un argumento positivo. La función RND genera la misma secuencia de números al azar con cada RUN, salvo que se use RANDOMIZE. (Ver Secciones 2.53 y 3.30.)
6. Se han modificado las reglas para imprimir (PRINT) números de precisión simple y doble. (Ver Sección 2.49.)
7. Se asigna dinámicamente espacio de serie y el primer argumento de una sentencia CLEAR de dos argumentos fija el fin de la memoria. El segundo argumento fija el total de espacio de apilamiento. (Ver Sección 2.4.)
8. La respuesta a INPUT con demasiados o muy pocos ítems, o con un tipo erróneo de valor (numérico en lugar de serie, etc.), o con un retorno de carro, imprime el mensaje "?Redo from start". No se asignan los valores de entrada hasta tanto no se dé una respuesta aceptable.
9. PRINT USING cuenta con dos nuevos caracteres de formato de campo. Se usa un ampersead en los campos de serie de longitud variable; un subrayado significa un caracter literal en una serie de formato.
10. Si la expresión provista con la sentencia WIDTH fuera 255, el BASIC usa un ancho de línea "infinito", esto es, no inserta retornos de carro. Puede usarse WIDTH LPRINT para fijar el ancho de línea en la impresora de líneas. (Ver Sección 2.66.)
11. El signo "at" y el subrayado dejan de usarse como caracteres de edición.
12. Los nombres de variables son significativos hasta 40 caracteres y pueden contener incluidas palabras reservadas. Sin embargo, las palabras reservadas deben ahora delimitarse mediante espacios. A fin de mantener la compatibilidad con las versiones anteriores del BASIC, se insertarán automáticamente espacios entre palabras reservadas y nombres de variables contiguos. ADVERTENCIA: Esta inserción de espacios puede truncar el fin de una línea si la longitud de ésta se aproximara a los 255 caracteres.
13. Los programas BASIC pueden resguardarse en un formato binario protegido. (Ver SAVE, Sección 2.60.)

CP/M e ISIS-II BASIC-80

En CP/M e ISIS-II BASIC-80, edición 5.0, se ha hecho un número de agregados a la posibilidad de entrada/salida en disco:

1. Después de una sentencia GET, puede hacerse que INPUT≠ y LINE INPUT≠ lean caracteres desde el buffer de archivo al azar. También pueden usarse PRINT≠, PRINT≠ USING y WRITE≠ para poner caracteres en el buffer de archivo al azar antes de una sentencia PUT.

   En el caso de WRITE≠, el BASIC-80 rellena el buffer con espacios hasta el retorno de carro. Todo intento de leer o de grabar más allá del fin del buffer provoca un error "Overflow de campo".

2. Al final de la línea de comando puede agregarse /S:⟨capacidad máx reg⟩ para fijar la capacidad máxima de registro a usar con los archivos al azar. La capacidad por defecto de registro es de 128 bytes.

Se ha agregado un nuevo dispositivo a la sentencia INPUT. Puede usarse una coma en lugar de un punto y coma después de la serie de indicación ("prompt") para suprimir el signo de interrogación. Por ejemplo, la sentencia INPUT "INGRESAR FECHA DE NACIMIENTO",B$ imprimirá la indicación sin signo de interrogación.

ANEXO B

ENTRADA/SALIDA EN DISCO BASIC-80

En este anexo se examinan los procedimientos de entrada/salida en disco para los usuarios que comienzan a utilizar el BASIC-80. Si fuera usted un principiante en el BASIC-80 ó tuviera errores afines a disco, lea estos procedimientos y ejemplos de programa para asegurarse que esté empleando correctamente las sentencias de disco.

Toda vez que en un comando o sentencia de disco se requiera un "nombrearchivo", use uno que se ajuste a los requerimientos de su sistema operativo. El sistema operativo CP/M agregará una extensión por defecto .BAS al nombrearchivo dado en un comando SAVE, RUN, MERGE o LOAD.

B.1 COMANDOS DE ARCHIVO DE PROGRAMAS

Veamos, a modo de repaso, los comandos y sentencias empleados en la manipulación de archivos de programas.

SAVE "nombrearchivo" [,A]	Graba en disco el programa que reside corrientemente en la memoria. La A opcional graba el programa como una serie de caracteres ASCII. (En caso contrario, el BASIC usa un formato binario comprimido.)
LOAD "nombrearchivo" [,R]	Carga el programa de disco en memoria. La R opcional opera el programa de inmediato. LOAD elimina siempre el contenido corriente de la memoria y cierra todos los archivos antes de cargar. Sin embargo, si se incluyera R, los archivos de datos abiertos siguen abiertos. Así, pues, pueden encadenarse o cargarse programas en secciones y acceder los mismos archivos de datos.
RUN "nombrearchivo" [,R]	RUN "nombrearchivo" carga el programa de disco en memoria y lo ejecuta. RUN elimina el contenido corriente de la memoria y cierra todos los archivos antes de cargar el programa. Sin embargo, si se incluyera la opción R, todos los archivos de datos abiertos siguen abiertos.
MERGE "nombrearchivo"	Carga el programa de disco en memoria pero no elimina el contenido corriente de la memoria. Los números de línea de programa del disco se intercalan con los números de línea de la memoria. Si dos líneas tuvieran el mismo número, sólo se resguarda la línea del programa en disco. Después de un comando MERGE, el programa "intercalado" reside en memoria y el BASIC vuelve a nivel de comando.
KILL "nombrearchivo"	Elimina el archivo del disco. "nombrearchivo" puede ser un archivo de programa o un archivo de datos secuencial o de acceso al azar.
NAME	Para modificar el nombre de un archivo en disco ejecútese la sentencia NAME "archivo anterior" AS "archivo nuevo". NAME puede usarse con archivos de programa, al azar o secuenciales.

B.2 ARCHIVOS PROTEGIDOS

Si desea resguardar un programa en un formato codificado en binario, use la opción "Protect" con el comando SAVE. Por ejemplo:

```
SAVE "MYPROG",P
```

Un programa resguardado de esta manera no puede listarse o editarse.

B.3 ARCHIVOS DE DATOS EN DISCO - ENTRADA/SALIDA SECUENCIAL Y AL AZAR

Dos son los tipos de archivos de datos en disco que un programa BASIC-80 puede crear y acceder: secuenciales y de acceso al azar.

B.3.1 **Archivos Secuenciales**

Los archivos secuenciales son más fáciles de crear que los archivos al azar, pero su flexibilidad y velocidad se ven limitadas al accederse los datos. Los datos que se graban en un archivo secuencial se almacenan un ítem tras el otro (secuencialmente), en el orden en que se transmiten, leyéndose de la misma manera.

Las sentencias y funciones que se usan con los archivos secuenciales son éstas:

OPEN	PRINT≠	INPUT≠	WRITE≠
	PRINT≠ USING	LINE INPUT≠	
CLOSE	EOF	LOC	

Para crear un archivo secuencial y acceder sus datos son necesarios los siguientes pasos de programa:

1. OPEN el archivo en modalidad "0"	OPEN "O",≠1,"DATA"
2. Grabar datos en el archivo usando la sentencia PRINT≠. (Puede usarse también WRITE≠).	PRINT≠1,A$;B$;C$
3. Para acceder los datos del archivo debe usted cerrarlo y abrirlo nuevamente en modalidad "I".	CLOSE≠1 OPEN "I",≠1,"DATA"
4. Usar la sentencia INPUT≠ para leer los datos del archivo secuencial en el programa.	INPUT≠1,X$,Y$,Z$

El programa B-1 es un programa breve que crea un archivo secuencial, "DATA", a partir de información que usted ingresa en la terminal.

```
25 IF N$="TERMINE" THEN END
20 INPUT "NOMBRE";N$
10 OPEN "0",≠1,"DATA"
30 INPUT "DEPARTAMENTO";D$
40 INPUT "FECHA DE INGRESO";H$
50 PRINT≠1,N$;",";D$;",";H$
60 PRINT:GOTO 20
RUN
NOMBRE? RATON MICKEY
DEPARTAMENTO: AYUDAS AUDIOVISUALES
FECHA DE INGRESO? 01/12/72
NOMBRE? SHERLOCK HOLMES
DEPARTAMENTO? INVESTIGACION
FECHA DE INGRESO? 12/03/65
NOMBRE? EBENEZER SCROOGE
DEPARTAMENTO? CONTABLE
FECHA DE INGRESO? 04/27/78
NOMBRE? SUPERMAN
DEPARTAMENTO? MANTENIMIENTO
FECHA DE INGRESO? 08/16/78
NOMBRE? etc.
```

PROGRAMA B-1 – CREAR UN ARCHIVO DE DATOS SECUENCIAL

Veamos ahora el programa B-2. Accede el archivo "DATA" que ha creado el programa B-1 y representa el nombre de todos quienes hayan ingresado en 1978.

```
10 OPEN "I",≠1,"DATA"
20 INPUT≠,N$,D$,H$
30 IF RIGHT$(H$,2)="78" THEN PRINT N$
```

```
40 GOTO 20
RUN
EBENEZER SCROOGE
SUPERMAN
Input past end in 20
Ok
```

PROGRAMA B-2 – ACCESO DE UN ARCHIVO SECUENCIAL

El programa B-2 lee secuencialmente cada uno de los ítems del archivo. Una vez leídos todos los datos, la línea 20 provoca un error "Input past end". Para evitarlo, insértese la línea 15 que usa la función EOF para probar el fin del archivo:

```
15 IF EOF(1) THEN END
```

y modifique la línea 40 a GOTO 15.

Un programa que crea un archivo secuencial puede asimismo grabar datos con formato en el disco con la sentencia PRINT≠ USING. Por ejemplo, podría usarse la sentencia

```
PRINT≠1,USING"≠≠≠≠.≠≠,";A,B,C,D
```

para grabar datos numéricos en disco sin delimitadores explícitos. La coma al final de la serie de formato sirve para separar los ítems en el archivo en disco.

La función LOC, cuando se la emplea con un archivo secuencial, devuelve el número de sectores que se han grabado en o leído desde el archivo desde que éste fue abierto. Un sector es un bloque de datos de 128 bytes.

B.3.1.1 Agregado de Datos a un Archivo Secuencial

Si tuviera usted un archivo secuencial residiendo en disco y posteriormente quisiera agregarle más datos a partir de su fin, no puede abrir simplemente el archivo en modalidad "0" e iniciar la grabación de datos. Tan pronto abre usted un archivo secuencial en modalidad "0" destruye su contenido corriente. Puede utilizarse el siguiente procedimiento para agregar datos en un archivo existente llamado "NOMBRES".

1. OPEN "NOMBRES" en modalidad "I".
2. OPEN un segundo archivo llamado "COPIA" en modalidad "0".
3. Leer los datos de "NOMBRES" y grabarlos en "COPIA".
4. CLOSE "NOMBRES" y luego KILL.
5. Grabar la nueva información en "COPIA".
6. Nombrar "COPIA" como "NOMBRES" y CLOSE.
7. En disco hay ahora un archivo llamado "NOMBRES" que incluye todos los datos anteriores más los nuevos que acaba usted de agregar.

El programa B-3 ilustra esta técnica. Puede usarse para crear o agregar en un archivo llamado NOMBRES. Este programa también ilustra el uso de LINE INPUT≠ para leer series con comas incluidas desde el archivo en disco. Recuerde, LINE INPUT≠ leerá caracteres del disco hasta detectar un retorno de carro (no para por comillas o comas) o luego de haber leído 255 caracteres.

```
10 ON ERROR GOTO 2000
20 OPEN "I",≠1,"NOMBRES"
30 REM SI EXISTE EL ARCHIVO, GRABARLO EN "COPIA"
40 OPEN "0",≠2,"COPIA"
50 IF EOF(1) THEN 90
60 LINE INPUT≠1,A$
70 PRINT≠2,A$
```

```
80 GOTO 50
90 CLOSE #1
100 KILL "NOMBRES"
110 REM AGREGAR NUEVAS ENTRADAS AL ARCHIVO
120 INPUT "NOMBRE";N$
130 IF N$="" THEN 200 'RETORNO DE CARRO SALIDA DEL LAZO DE ENTRADA
140 LINE INPUT "DIRECCION?";A$
150 LINE INPUT "NACIMIENTO?";B$
160 PRINT#2,N$
170 PRINT#2,A$
180 PRINT#2,B$
190 PRINT:GOTO 120
200 CLOSE
205 REM MODIFICAR NOMBREARCHIVO A "NOMBRES"
210 NAME "COPIA" AS "NOMBRES"
2000 IF ERR=53 AND ERL=20 THEN OPEN "O",#2,"COPIA";RESUME 120
2010 ON ERROR GOTO 0
```

PROGRAMA B-3 – AGREGADO DE DATOS A UN ARCHIVO SECUENCIAL

La rutina de desvío por errores de la línea 2000 detecta un error "No existe el archivo" en la línea 20. Si esto aconteciera, se omiten las sentencias que copian el archivo y se crea "COPIA" tal como si fuera un archivo nuevo.

B.3.2 **Archivos al Azar**

La creación y el acceso de los archivos al azar requiere más pasos de programa que los archivos secuenciales, pero del uso de archivos al azar derivan ventajas. Una de ellas es que los archivos al azar requieren menos espacio en disco puesto que el BASIC los almacena en formato binario empaquetado. (Un archivo secuencial se almacena como una serie de caracteres ASCII.)

La mayor ventaja de los archivos al azar radica en que los datos pueden accederse aleatoriamente, es decir, en cualquier lugar del disco; no es necesario leer toda la información, tal como en los archivos secuenciales. Esto es posible porque la información se almacena y accede en unidades llamadas registros y cada uno de los registros tiene un número.

Las sentencias y funciones que se usan con los archivos al azar son:

OPEN	FIELD	LSET/RSET	GET
PUT	CLOSE	LOC	
MKI$	CVI		
MKS$	CVS		
MKD$	CVD		

B.3.2.1 **Creación de un Archivo al Azar**

Para crear un archivo al azar son necesarios los siguientes pasos de programa:

1. OPEN el archivo para acceso al azar (modalidad "R"). Este ejemplo especifica una longitud de registro de 32 bytes. Si se omitiera la longitud del registro, el valor por defecto es 128 bytes.

```
OPEN "R",#1,"FILE",32
```

2. Usar la sentencia FIELD para asignar espacio en el buffer al azar para las variables que se grabarán en el archivo al azar.

```
FIELD #1 20 AS N$,
 4 AS A$, 8 AS P$
```

3. Usar LSET para transferir los datos al buffer al azar. Los valores numéricos han de convertirse en series al colocarse en el buffer. Para hacer esto, use las funciones "make": MKI$ para convertir un valor entero en una serie; MKS$ para un valor de precisión simple y MKD$ para un valor de precisión doble.

```
LSET N$=X$
LSET A$=MKS$(AMT)
LSET P$=TEL$
```

4. Grabar los datos del buffer en el disco usando la sentencia PUT.

```
PUT #1,CODE%
```

Ver el programa B-4. Toma la información ingresada en la terminal y la graba en un archivo al azar. Toda vez que se ejecuta la sentencia PUT se graba un registro en el archivo. El código de dos dígitos que se ingresa en la línea 30 se convierte en el número del registro.

NOTA: No use una variable de serie FIELD en una sentencia INPUT o LET. Esto hace que el indicador ("pointer") de esa variable apunte al espacio de serie en lugar del buffer de archivo al azar.

```
10 OPEN "R" #1, "FILE"
20 FIELD #1,20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "CODIGO DOS DIGITOS";CODE%
40 INPUT "NOMBRE";X$
50 INPUT "IMPORTE";AMT
60 INPUT "TELEF";TEL$:PRINT
70 LSET N$=X$
80 LSET A$=MKS$(AMT)
90 LSET P$=TEL$
100 PUT #1,CODE%
110 GOTO 30
```

PROGRAMA B-4 – CREACION DE UN ARCHIVO AL AZAR

B.3.2.2 **Acceso a un Archivo al Azar**

Para acceder un archivo al azar son necesarios los siguientes pasos de programa:

1. OPEN el archivo en modalidad "R".

```
OPEN "R",#1,"FILE",32
```

2. Usar la sentencia FIELD para asignar espacio en el buffer al azar para las variables que se leerán desde el archivo.

```
FIELD #1 20 AS N$,
      4 AS A$, 8 AS P$
```

NOTA: En un programa que ejecute tanto entrada como salida en el mismo archivo al azar puede usted con frecuencia usar solamente una sentencia OPEN y una FIELD.

3. Usar la sentencia GET para transferir el registro deseado al buffer al azar.

```
GET #1,CODE%
```

4. El programa puede ahora acceder los datos del buffer. Los valores numéricos deben reconvertirse a números usando las funciones de "conversión": CVI para enteros, CVS para valores de precisión simple y CVD para valores de precisión doble.

```
PRINT N$
PRINT CVS(A$)
```

El programa B-5 accede al archivo al azar "FILE" creado en el programa B-4. Ingresando el código de tres dígitos en la terminal, la información asociada con ese código se lee en el archivo y se representa:

```
10 OPEN "R",#1,"FILE"
20 FIELD #1, 20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "CODIGO DE 2 DIGITOS";CODE%
40 GET #1,CODE%
50 PRINT N$
```

```
60 PRINT USING "$$≠≠≠.≠≠";CVS(A$)
70 PRINT P$:PRINT
80 GOTO 30
```

PROGRAMA B-5 – ACCESO A UN ARCHIVO AL AZAR

La función LOC, en los archivos al azar, devuelve el "número corriente de registro". El número corriente de registro es uno más el último número de registro usado en una sentencia GET o PUT. Por ejemplo, la sentencia

```
IF LOC(1)<50 THEN END
```

termina la ejecución del programa si el número de registro corriente del archivo ≠1 fuera mayor que 50.

El programa B-6 es un programa de inventario que ilustra el acceso al azar de un archivo. En este programa se usa el número de registro como número de matrícula y se presupone que las existencias contendrán no más de 100 diferentes matrículas. Las líneas 900-960 inicializan el archivo de datos grabando CHR$(255) como el primer caracter de cada uno de los registros. Esto se usa luego (línea 270 y línea 500) para determinar si ya existe una entrada para esa matrícula.

Las líneas 130-220 representan las diferentes funciones de inventario que cumple el programa. Cuando usted ingresa el número de función deseada la línea 230 bifurca a la pertinente subrutina.

PROGRAMA B-6 – INVENTARIO

```
120 OPEN "R",≠1,"DAT.INVEN",39
125 FIELD≠1,1 AS F$, 30 AS D$, 2 AS Q$, 2 AS R$, 4 AS P$
130 PRINT:PRINT "FUNCIONES:":PRINT
135 PRINT 1, "INICIALIZAR ARCHIVO"
140 PRINT 2, "CREAR NUEVA ENTRADA"
150 PRINT 3, "REPRESENTAR EXISTENCIA UNA MATRICULA"
160 PRINT 4, "SUMAR A STOCK"
170 PRINT 5, "RESTAR DE STOCK"
180 PRINT 6, "REPRESENTAR ITEMS POR DEBAJO NIVEL DE REPOSICION"
220 PRINT:PRINT:INPUT"FUNCION";FUNCTION
225 IF (FUNCTION<1)OR(FUNCTION>6) THEN PRINT "NUMERO FUNCION ERRONEO": GOTO 130
230 ON FUNCTION GOSUB 900,250,390,480,560,680
240 GOTO 220
250 REM CONSTRUIR NUEVA ENTRADA
260 GOSUB 840
270 IF ASC(F$)<>255 THEN INPUT "SOBREGRABAR";A$:IF A$<>"Y" THEN RETURN
280 LSET F$=CHR$(0)
290 INPUT "DESCRIPCION";DESC$
300 LSET D$=DESC$
310 INPUT "CANTIDAD EN STOCK";Q%
320 LSET Q$=MKI$(Q%)
330 INPUT "NIVEL REPOSICION";R%
340 LSET R$=MKI$(R%)
350 INPUT "PRECIO UNITARIO";P
360 LSET P$=MKS$(P)
370 PUT≠1,PART%
380 RETURN
390 REM REPRESENTAR ENTRADA
```

```
400 GOSUB 840
410 IF ASC(F$)=255 THEN PRINT "ENTRADA NULA":RETURN
420 PRINT USING "MATRICULA ###";PART%
430 PRINT D$
440 PRINT USING "CANTIDAD EN EXISTENCIA ####";CVI(Q$)
450 PRINT USING "NIVEL REPOSICION #####";CVI(R$)
460 PRINT USING "PRECIO UNITARIO $$##.##";CVS(P$)
470 RETURN
480 REM SUMAR A STOCK
490 GOSUB 840
500 IF ASC(F$)=255 THEN PRINT "ENTRADA NULA":RETURN
510 PRINT D$:INPUT "CANTIDAD A SUMAR";A%
520 Q%=CVI(Q$)+A%
530 LSET Q$=MKI$(Q%)
540 PUT#1,PART%
550 RETURN
560 REM RETIRAR DE STOCK
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "ENTRADA NULA":RETURN
590 PRINT D$
600 INPUT "CANTIDAD A RESTAR";S%
610 Q%=CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "SOLO";Q%;"EN STOCK":GOTO 600
630 Q%=Q%-S%
640 IF Q%=<CVI(R$) THEN PRINT "CANTIDAD AHORA";Q%;"NIVEL REPOSICION";CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT#1,PART%
670 RETURN
680 REM REPRESENTAR ITEMS POR DEBAJO NIVEL REPOSICION
690 FOR I=1 TO 100
710 GET#1,I
720 IF CVI(Q$)<CVI(R$) THEN PRINT D$;"CANTIDAD";CVI(Q$) TAB(50)
    "NIVEL REPOS";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "MATRICULA";PART%
850 IF (PART%<1)OR(PART%>100) THEN PRINT "NUMERO MATRIC ERRONEO":GOTO 840
    ELSE GET#1,PART%:RETURN
890 END
900 REM INICIALIZAR
910 INPUT "ESTA SEGURO";B$:IF B$<>"Y" THEN RETURN
920 LSET F$=CHR$(255)
930 FOR I=1 TO 100
940 PUT#1,I
950 NEXT I
960 RETURN
```

ANEXO C

SUBRUTINAS EN LENGUAJE ASSEMBLY

Todas las versiones del BASIC-80 cuentan con provisiones para la interfase con subrutinas en lenguaje Assembly. La función USR permite llamar subrutinas en lenguaje Assembly de la misma manera en que se llaman las funciones intrínsecas del BASIC.

NOTA: Las direcciones de las rutinas DEINT, GIVABF, MAKINT y FRCINT se almacenan en ubicaciones que deben suministrarse individualmente para las diferentes implementaciones del BASIC.

C.1. ASIGNACION DE MEMORIA

Debe separarse espacio de memoria para una subrutina en lenguaje Assembly antes de que ésta pueda cargarse. Durante la inicialización, ingrese la ubicación más alta de memoria menos el total de memoria necesario para la(s) subrutina(s) en lenguaje Assembly. El BASIC utiliza toda la memoria disponible desde su posición inicial, de modo que sólo pueden separarse las ubicaciones superiores para las subrutinas del usuario.

Al llamarse una subrutina en lenguaje Assembly, se establece el indicador ("pointer") de pila para 8 niveles (16 bytes) de almacenamiento de pila. Si se necesitara más espacio de pila, puede resguardarse la pila del BASIC y establecerse una nueva para uso de la subrutina en lenguaje Assembly. Sin embargo, antes de volver de la subrutina, debe restaurarse la pila del BASIC

La subrutina en lenguaje Assembly puede cargarse en memoria por medio del monitor del sistema o de la sentencia BASIC POKE o (si el usuario tuviera el paquete MACRO-80 ó FORTRAN-80) pueden compaginarse las rutinas con MACRO-80 y cargarse usando LINK-80.

C.2 LLAMADOS DE FUNCION USR - BASIC 8K

La dirección inicial de la subrutina en lenguaje Assembly debe almacenarse en USRLOC, una ubicación de memoria de dos bytes que se provee individualmente con las diferentes implementaciones del BASIC-80. Con el BASIC 8K, la dirección inicial puede ser POKE en USRLOC. Almacenar primero el byte de orden inferior, seguido por el byte de orden superior.

La función USR llamará a la rutina cuya dirección esté en USRLOC. Inicialmente USRLOC contiene la dirección de ILLFUN, la rutina que da el error "Llamado inválido de función". Por lo tanto, si se llamara USR sin modificar la dirección que está en USRLOC, resulta un error "Llamado inválido de función".
El formato de un llamado de función USR es

USR (argumento)

en donde argumento es una expresión numérica. Para obtener el argumento, la subrutina en lenguaje Assembly debe llamar a la rutina DEINT. DEINT coloca el argumento en el par de registros D,E como un entero de 2 bytes con complemento a 2. (Si el argumento no estuviera dentro de los límites —32768 a 32767 resulta un error "Llamado inválido de función").

Para retransferir el resultado desde una subrutina en lenguaje Assembly, cargue el valor en el par de registros (A,B) y llame a la rutina GIVABF. Si no se llamara GIVABF, USR(X) devuelve X. Para volver al BASIC, la subrutina en lenguaje Assembly debe ejecutar una instrucción RET.

Por ejemplo, ésta es una subrutina en lenguaje Assembly que multiplica el argumento por 2:

```
USRSUB: CALL DEINT      ;poner argumento en D,E
        XCHG            ;transferir argumento a H,L
        DAD H           ;H,L=H,L+H,L
        MOVE A,H        ;transferir resultado a A,B
        MOVE B,L
```

```
JMP GIVABF      ;transferir resultado y RETurn
```

Obsérvese que de esta rutina se obtendrán resultados válidos dentro de los límites $-16384 = x = 16383$. La instrucción JMP GIVABF tiene el mismo efecto que:

```
CALL GIVABF
RET
```

Para devolver valores adicionales al programa, cárguelos en memoria y léalos con la función PEEK.

Varios son los métodos por los cuales un programa puede llamar a más de una rutina USR. Por ejemplo, la dirección inicial de cada una de las rutinas puede ser POKE en USRLOC antes de cada llamado USR, o el argumento de USR podría ser un índice de una tabla de rutinas USR.

C.3 LLAMADOS DE FUNCION USR - BASIC AMPLIADO Y DISCO

En las versiones Ampliado y Disco, el formato de la función USR es el siguiente:

USR [⟨dígito⟩] (argumento)

en donde ⟨dígito⟩ está entre 0 y 9 y el argumento es cualquier expresión numérica o de serie. ⟨dígito⟩ especifica a qué rutina USR se está llamando, y se corresponde con el dígito provisto en la sentencia DEF USR para esa rutina. Si se omitiera ⟨dígito⟩, se presupone USR0. La dirección dada en la sentencia DEF USR determina la dirección inicial de la subrutina.

Al hacerse el llamado de función USR, el registro A contiene un valor que especifica el tipo de argumento que se dio. El valor de A puede ser uno de los siguientes:

Valor de A	Tipo de Argumento
2	Entero de dos bytes (complemento a dos)
3	Serie
4	Número de punto flotante de precisión simple
8	Número de punto flotante de precisión doble

Si el argumento fuese un número, el par de registros (H,L) apunta al Acumulador de Punto Flotante (FAC) en el que se almacena el argumento.

Si el argumento fuera un entero:

FAC-3 contiene los 8 bits inferiores del argumento, y

FAC-2 contiene los 8 bits superiores del argumento.

Si el argumento fuese un número de punto flotante de preecisión simple:

FAC-3 contiene los 8 bits inferiores de la mantisa, y

FAC-2 contiene los 8 bits intermedios de la mantisa, y

FAC-1 contiene los 7 bits superiores de la mantisa, suprimiéndose el 1 a la izquierda (implícito). El bit 7 es el signo del número (0 = positivo, 1 = negativo). FAC es el exponente menos 128, y el punto binario está a la izquierda del bit más significativo de la mantisa.

Si el argumento fuese un número de punto flotante de prcisión doble:

FAC-7 a FAC-4 contienen cuatro bytes más de mantisa

(FAC-7 contiene los 8 bits inferiores).

Si el argumento fuese una serie, el par de registros (D,E) apunta a 3 bytes llamados "descriptor de serie". El byte 0 del descriptor de serie contiene la longitud de la serie (0 a 255). Los bytes 1 y 2, respectivamente, son los 8 bits inferiores y superiores de la dirección inicial de la serie en el espacio de serie.

PRECAUCION: Si el argumento fuese un literal de serie del programa, el descriptor de serie apuntará al texto del programa. Tenga cuidado de no alterar o destruir su programa de esta manera. Para evitar resultados impronosticables, agregue +" " al literal de serie del programa. Ejemplo:

```
A$ = "BASIC-80"+" "
```

Esto copiará el literal de serie en el espacio de serie e impedirá la alteración del texto del programa durante un llamado de subrutina.

Usualmente, el valor que devuelve una función USR es del mismo tipo (entera, de serie, de precisión simple o de precisión doble) que el argumento que le fue transferido. Sin embargo, el llamado de la rutina MAKINT devuelve el

entero de (H,L) como el valor de la función, forzando el valor que devuelve la función a entero. Para ejecutar MAKINT, utilice la siguiente secuencia para volver desde la subrutina.

```
PUSH    H       ;resguardar valor a devolver
LHLD    xxx     ;obtener dirección de la rutina MAKINT
XTHL            ;resguardar retorno en pila y
                ;recuperar (H,L)
RET             ;retorno
```

Asimismo, el argumento de la función, sea cual fuere su tipo, puede forzarse a entero llamando a la rutina FRCINT para obtener el valor entero del argumento de (H,L). Ejecutar la siguiente rutina:

```
LXI     H       ;obtener dirección de subrutina
                ;continuación
PUSH    H       ;colocar en pila
LHLD    xxx     ;obtener direcciôn de FRCINT
PCHL
SUB1:.....
```

C.4 SENTENCIA CALL

Pueden también llamarse funciones del usuario del BASIC-80 Ampliado y Disco con la sentencia CALL. La secuencia llamadora que se utiliza es la misma que la de los compiladores FORTRAN, COBOL y BASIC de Microsoft.

Una sentencia CALL sin argumentos genera una simple instrucción "CALL". La correspondiente subrutina deberá volver a través de una "RET". (CALL y RET son códigos de operación 8080; consultar detalles en un manual 8080). Una CALL de subrutina con argumentos resulta en una secuencia llamadora algo más compleja. Por cada argumento de la lista de argumentos CALL se transfiere un parámetro a la subrutina. Ese parámetro es la dirección del byte inferior del argumento. Por lo tanto, los parámetros ocupan siempre dos bytes cada uno, sea cual fuere el tipo.
El método para transferir parámetros depende del número de parámetros a transferir:

1. Si el número de parámetros fuese menor que o igual a 3, se transfieren en los registros. El parámetro 1 lo será en HL, el 2 en DE (de estar presente) y el 3 en BC (de estar presente).
2. Si el número de parámetros fuese mayor que 3, se transfieren de la siguiente manera:
   1. El parámetro 1 en HL.
   2. El parámetro 2 en DE.
   3. Los parámetros 3 a n en un bloque de datos contiguo. BC apuntará al byte inferior de este bloque de datos (esto es, al byte inferior del parámetro 3).

Obsérvese que, con este esquema, la subrutina debe conocer cuántos parámetros esperar a fin de hallarlos. Recíprocamente, el programa llamador es responsable de transferir el número correcto de parámetros. No se verifica el número o tipo correcto de los parámetros.

Si la subrutina esperara más de 3 parámetros y necesitara transferirlos a un área local de datos, existe una subrutina del sistema que cumplirá esta transferencia. Esta rutina de transferencia de argumentos se llama $AT (ubicada en la biblioteca FORTRAN FORLIB.REL) y se llama con HL apuntando al área local de datos, con BC apuntando al tercer parámetro y con A conteniendo el número de argumentos a transferir (esto es, el número total de argumentos menos 2). La subrutina es responsable de resguardar los primeros dos parámetros antes de llamar a $AT.

Por ejemplo, si una subrutina espera 5 parámetros, podría ser algo así:

```
SUBR: SHLD    P1      ;RESGUARDAR PARAMETRO 1
      XCHG
      SHLD    P2      ;RESGUARDAR PARAMETRO 2
      MVI     A,3     ;Nº DE PARAMETROS RESTANTES
      LXI     H,P3    ;INDICADOR DEL AREA LOCAL
      CALL    $AT     ;TRANSFERIR LOS OTROS 3 PARAMETROS
      .
      .
      .
```

```
        (cuerpo de la subrutina)
        .
        .
        .
        RET                 ;RETORNO AL LLAMADOR
P1:     DS      2           ;ESPACIO PARA EL PARAMETRO 1
P2:     DS      2           ;ESPACIO PARA EL PARAMETRO 2
P3:     DS      6           ;ESPACIO PARA LOS PARAMETROS 3-5
```

Sigue un listado de la rutina AT$ de transferencia de argumentos.

```
00100 ;         TRANSFERENCIA DE ARGUMENTOS
00200 ;(B,C)    APUNTA AL TERCER PARAMETRO
00300 ;(H,L)    APUNTA AL ALMACENAMIENTO LOCAL PARA EL PARAMETRO 3
00400 ;(A)      CONTIENE EL Nº DE PARAMETROS A XFER(TOTAL—2)
00500
00600
00700           ENTRY   $AT
00800 $AT:              XCHG    ;RESGUARDAR (H,L) en (D,E)
00900           H,B     MOV
01000           L,C     MOV     ;(H,L) = PTR A PARAMETROS
01100 AT1:      C,M     MOV
01200           H       INX
01300           B,M     MOV
01400           H       INX     ;(B,C) = DIRECCION PARAM
01500                   XCHG    ;(H,L) APUNTA A ALMACEN LOCAL
01600           M,C     MOV
01700           H       INX
01800           M,B     MOV
01900           H       INX     ;ALMACENAR PARAM EN AREA LOCAL
02000                   XCHG    ;DESDE VUELTA A AT1
02100           A       DCR     ;TRANSFERIDOS TODOS LOS PARAMETROS?
02200           AT1     JNZ     ;NO, COPIAR MAS
02300                   RET     ;SI, VOLVER
```

Al acceder parámetros en una subrutina no olvide que existen **indicadores** a los argumentos reales transferidos.

NOTA: Corre por cuenta del programador comprobar si los argumentos del programa llamador concuerdan en **número, tipo** y **longitud** con los parámetros que espera la subrutina. Vale esto para las subrutinas BASIC y también para las escritas en lenguaje Assembly.

C.5 INTERRUPCIONES

Pueden escribirse subrutinas en lenguaje Assembly para atender interrupciones. Todas las rutinas de atención de interrupciones resguardarán la pila, el registro A-L y la PSW. Las interrupciones deberán reactivarse siempre antes de volver de la subrutina, puesto que una interrupción desactiva automáticamente todas las otras una vez recibida. El usuario deberá conocer qué vectores de interrupción están libres en la versión particular del BASIC que se le haya provisto. Nota para los usuarios CP/M: En CP/M BASIC, todos los vectores de interrupción están libres.

ANEXO D

BASIC-80 CON EL SISTEMA OPERATIVO CP/M

La versión CP/M del BASIC-80 (MBASIC) se provee en un diskette de densidad simple 3740 de capacidad estándar. El nombre del archivo es MBASIC.COM. (Se recomienda un sistema CP/M de 28K o más).
Para correr MBASIC, active CP/M y digite lo siguiente:

A>MBASIC ⟨retorno de carro⟩

El sistema responderá:

xxxx Bytes Free
BASIC-80 Version 5.0
(CP/M Version)
Copyright 1978 © by Microsoft
Created: dd-mmm-yy
Ok

MBASIC es igual al BASIC-80 en Disco, tal como se describe en este manual, con las siguientes excepciones:

D.1 INICIALIZACION

El diálogo de inicialización ha sido reemplazado por un conjunto de opciones que se colocan después del comando MBASIC al CP/M. El formato de la línea de comando es:

A>MBASIC [⟨nombrearchivo⟩] [/F: ⟨número archivos⟩] [/M: ⟨ubicación más alta de memoria⟩]

Si estuviera presente ⟨nombrearchivo⟩, MBASIC continúa como si se hubiera digitado un comando RUN ⟨nombrearchivo⟩ después de completarse la inicialización. Se usa una extensión por defecto de .BAS si no se proveyera ninguna y el nombrearchivo fuera de menos de 9 caracteres de longitud. Esto permite que los programas BASIC ejecuten en modalidad "batch" usando el medio SUBMIT de CP/M. Tales programas incluirán una sentencia SYSTEM (ver más adelante) para volver al CP/M cuando hayan terminado, permitiendo ejecutar al siguiente programa de la corriente "batch".

Si /F: ⟨número archivos⟩ estuviera presente, fija el número de archivos de datos en disco que pueden abrirse en cualquier momento de la ejecución de un programa BASIC. Todo bloque de datos de archivo asignado de esta manera requiere 166 bytes de memoria. Si se omitiera la opción /F, el número de archivo será 3 por defecto.

La opción /M: ⟨ubicación más alta de memoria⟩ fija la ubicación más alta de memoria que usará MBASIC. En algunos casos es deseable fijar el total de memoria bien por debajo de FDOS de CP/M para reservar espacios a las subrutinas en lenguaje Assembly. En todos los casos, ⟨ubicación más alta de memoria⟩ deberá estar por debajo de la iniciación de FDOS (cuya dirección está contenida en las ubicaciones 6 y 7). Si se omitiera la opción /M, se usa toda la memoria hasta la iniciación de FDOS.

NOTA: Número archivos y ubicación más alta de memoria son números que pueden ser decimales, octales (precedidos por &O) o hexadecimales (precedidos por &H).

Ejemplos:

A>MBASIC PAYROLL.BAS	Usa toda la memoria y 3 archivos, carga y ejecuta PAYROLL.BAS.
A>MBASIC INVENT/F:6	Usa toda la memoria y 6 archivos, carga y ejecuta INVENT.BAS.
A>MBASIC /M:32768	Usa los primeros 32K de memoria y 3 archivos.
A>MBASIC DATACK/F:2/M:&H9000	Usa los primeros 36K de memoria, 2 archivos y ejecuta DATACK.BAS.

D.2 ARCHIVOS EN DISCO

Los nombrearchivo en disco siguen las convencionales normales de nombres CP/M. Todos los nombrearchivo pue-

den incluir A; o B: como primeros dos caracteres para especificar una unidad de disco; en caso contrario se presupone la unidad corrientemente seleccionada. En los comandos LOAD, SAVE, MERGE y RUN nombrearchivo se usa una extensión por defecto de .BAS si en el nombrearchivo no aparece ningún "." y el nombrearchivo tuviera menos de 9 caracteres de longitud.

D.3 **COMANDO FILES**

Formato: FILES ⟨nombrearchivo⟩

Propósito: Imprimir los nombres de los archivos que residen en el disco corriente.

Observaciones: Si se omitiera ⟨nombrearchivo⟩, se listarán todos los archivos de la unidad corrientemente seleccionada. ⟨nombrearchivo⟩ es una fórmula de serie que pueden contener signos de interrogación (?) para concordar cualquier caracter del nombrearchivo o extensión. Un asterisco (*) como primer caracter del nombrearchivo o extensión concordará cualquier archivo o extensión.

Ejemplos:
```
FILES
FILES "*.BAS"
FILES "B:*.*"
FILES "TEST?.BAS"
```

D.4 **COMANDO RESET**

Formato: RESET

Propósito: Cerrar todos los archivos en disco y grabar la información de directorio en un diskette antes de ser retirado de una unidad de disco.

Observaciones: Ejecute siempre un comando RESET antes de retirar un diskette de una unidad de disco. De no ser así, al usarse nuevamente el diskette, no tendrá la información corriente de directorio grabada en la pista de directorio.

RESET cierra todos los archivos abiertos en todas las unidades y graba la pista de directorio en todo diskette con archivos abiertos.

D.5 **FUNCION LOF**

Formato: LOF (⟨número archivo⟩)

Acción: Devuelve el número de registros presentes en la última extensión leída o grabada. Si el archivo no excediera una extensión (128 registros), LOF devuelve la longitud verdadera del archivo.

Ejemplo:
```
110 IF NUM%>LOF(1) THEN PRINT "ENTRADA INVALIDA"
```

D.6 **EOF**

Con CP/M, puede usarse la función EOF con archivos al azar. Si se diera GET más allá del fin del archivo, EOF devolverá —1. Esto puede usarse para hallar la capacidad de un archivo utilizando una búsqueda binaria u otro algoritmo.

D.7 **VARIOS**

1. CSAVE y CLOAD no están implementados.
2. Para volver a CP/M, use el comando o sentencia SYSTEM. SYSTEM cierra todos los archivos y luego realiza un arranque en caliente CP/M. Control-C vuelve siempre a MBASIC, no a CP/M.
3. FRCINT está en 103 hexadecimal y MAKINT en 105 hexadecimal. (Sumar 1000 hexadecimal para versiones ADDS, 4000 para versiones SBC CP/M).

ANEXO E

CONVERSION DE PROGRAMAS AL BASIC-80

Si tuviera usted programas que están escritos en un BASIC que no es el BASIC-80, antes de operarlos con éste quizá sean necesarios algunos ajustes menores. Siguen una serie de puntos específicos que observará al convertir programas BASIC.

E.1 DIMENSIONES DE SERIES

Elimine todas las sentencias que se usen para declarar la longitud de las series. Una sentencia tal como DIM A$(I,J) que dimensiona una matriz de serie para J elementos de longitud I deberá convertirse a la sentencia BASIC-80 DIM A$(J).

Algunos BASICs usan una coma o ampersand para concatenar series. Deben modificarse a un signo más, que es el operador de concatenación de series del BASIC-80.

En el BASIC-80 las funciones MID$, RIGHT$ y LEFT$ se usan para subíndices de series. Los formatos tales como A$(I) para acceder al Iº carácter de A$, o A$(I,J) para tomar un subíndice de A$ desde la posición I a la posición J, debe modificarse de la siguiente manera:

Otros BASIC	BASIC-80
X$=A$(I)	X$=MID$(A$,I,1)
X$=A$(I,J)	X$=MID$(A$,I,J−I+1)

Si la referencia al subíndice está en el lado izquierdo de una asignación y se usara X$ para reemplazar los caracteres de A$, convertir de la siguiente manera:

Otros BASIC	BASIC-80 8K
A$(I)=X$	A$=LEFT$(A$,I−1)+X$+MID$(A$,I+1)
A$(I,J)=X$	A$=LEFT$(A$,I−1);X$;MID$(A$,J+1)
	BASIC-80 Ampliado y Disco
A$(I)=X$	MID$(A$,1,1)=X$
A$(I,J)=X$	MID$(A$,I,J−I+1)=X$

E.2 ASIGNACIONES MULTIPLES

Algunos BASICs permiten sentencias del formato

```
10 LET B=C=0
```

para igualar B y C a cero. El BASIC-80 interpretaría el segundo signo igual como un operador lógico y haría B igual a −1 si C igualó a cero. En lugar de esto, convierta la sentencia a dos sentencias de asignación:

```
10 C=O:B=0
```

E.3 SENTENCIAS MULTIPLES

Algunos BASICs emplean una barra inversa (\) para separar sentencias múltiples en una línea. Corrobore, con el BASIC-80, que todas las sentencias de una línea estén separadas con dos puntos (:).

E.4 FUNCIONES MAT

Los programas que usen las funciones MAT disponibles en algunos BASICs han de ser escritos nuevamente usando lazos FOR . . . NEXT para ejecutar adecuadamente.

ANEXO F

RESUMEN DE CODIGOS Y DE MENSAJES DE ERROR

	Número	Mensaje
NF	1	NEXT without FOR Una variable de una sentencia NEXT no corresponde a ninguna variable de sentencia FOR previamente ejecutada, sin concordancia.
SN	2	Syntax error Se detecta una línea que contiene alguna secuencia incorrecta de caracteres (como paréntesis sin concordar, comando o sentencia mal escrito, puntuación incorrecta, etc.).
RG	3	Return without GOSUB Se detecta una sentencia RETURN para la cual no existe una sentencia GOSUB previa, sin concordar.
OD	4	Out of data Se ejecuta una sentencia READ cuando en el programa restan sentencias DATA con datos no leídos.
FC	5	Illegal function call Se transfiere a una función matemática o de serie un parámetro que está fuera de límite. Puede también producirse un error FC como resultado de: 1. un subíndice negativo o irrazonablemente grande 2. un argumento negativo o cero con LOG 3. un argumento negativo para SQR 4. una mantisa negativa con un exponente no entero 5. un llamado a una función USR para el cual no se ha dado aún la dirección inicial 6. un argumento inadecuado para MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR u ON . . . GOTO.
OV	6	Overflow El resultado de un cálculo es demasiado grande para ser representado en el formato de números del BASIC-80. Si se produce "underflow" el resultado es cero y la ejecución continúa sin error.
OM	7	Our of memory Un programa es demasiado grande, tiene demasiados lazos FOR o GOSUBs, demasiadas variables o expresiones demasiado complicadas.
UL	8	Undefined line La referencia a una línea en una GOTO, GOSUB, IF . . . THEN . . . ELSE o DELETE es a una línea que no existe.
BS	9	Subscript out of range Se referencia a un elemento de la matriz con un subíndice que está fuera de las dimensiones de la matriz o con un número erróneo de subíndices.
DD	10	Redimensioned array Se dan dos sentencias DIM para la misma matriz o se da una sentencia DIM para una matriz luego de fijada para esa matriz la dimensión por defecto de 10.

Código	Número	Mensaje
/0	11	Division by cero En una expresión se detecta una división por cero o la operación de involución resulta en cero elevado a una potencia negativa. Como resultado de la división se proporciona infinitud de máquina con el signo del numerador, o se provee infinitud de máquina positiva como resultado de la involución y la ejecución continúa.
ID	12	Illegal direct Se ha ingresado una sentencia inválida en modalidad directa como un comando en modalidad directa.
TM	13	Type mismatch Se asigna un nombre de variable de serie a un valor numérico o viceversa; una función que espera un argumento numérico recibe un argumento de serie o viceversa.
OS	14	Out of string space Las variables de serie han hecho que el BASIC excediera el total de memoria libre restante. El BASIC asignará espacio de serie dinámicamente, hasta quedar sin memoria.
LS	15	String too long Se ha intentado crear una serie de más de 255 caracteres de longitud.
ST	16	String formula too complex Una expresión de serie es demasiado larga o demasiado compleja. La expresión deberá descomponerse en expresiones más pequeñas.
CN	17	Can't continue Se ha intentado continuar un programa que: 1. ha parado debido a un error 2. ha sido modificado durante una interrupción de la ejecución, o 3. no existe.
UF	18	Undefined user function Se ha llamado a una función USR antes de darse su definición (sentencia DEF).
		Versiones Ampliado y Disco Solamente
	19	No RESUME Se ingresa en una rutina de desvío por error mas no contiene ninguna sentencia RESUME.
	20	RESUME without error Se detecta una sentencia RESUME antes de ingresarse en una rutina de desvío por error.
	21	Unprintable error No se dispone de un mensaje de error para la condición de error que existe. Por lo común esto lo provoca un ERROR con un código de error sin definir.
	22	Missing operand Una expresión contiene un operador sin que le siga ningún operando.
	23	Line buffer overflow Se ha intentado dar entrada a una línea que tiene demasiados caracteres.
	26	FOR without NEXT Se ha detectado una FOR sin una NEXT que concuerde.
	29	WHILE without WEND Una sentencia WHILE no tiene una WEND que concuerde.
	30	WEND without WHILE Se ha detectado una WEND sin una WHILE que concuerde.
		Errores en Disco
	50	Field overflow

Código	Número	Mensaje
		Una sentencia FIELD está intentando asignar más bytes que los especificados para la longitud de registro de un archivo al azar.
	51	Internal error En el BASIC-80 Disco se ha producido un malfuncionamiento interno. Informar a Microsoft las condiciones en las cuales apareció el mensaje.
	52	Bad file number Una sentencia o un comando referencia a un archivo con un número de archivo que no está abierto o está fuera de los límites de números de archivo especificados en la inicialización.
	53	File not found Una sentencia LOAD, KILL u OPEN referencia a un archivo que no existe en el disco corriente.
	54	Bad file mode Se intenta usar PUT, GET o LOF con un archivo secuencial para cargar (LOAD) un archivo al azar o para ejecutar una OPEN con una modalidad de archivo que no es I, O o R.
	55	File already open Se emite una OPEN en modalidad de salida secuencial para un archivo que ya está abierto; o se da una KILL para un archivo que está abierto.
	57	Disk I/O error Se ha producido un error de entrada/salida en una operación de entrada/salida en disco. Es un error fatal, esto es, el sistema operativo no puede recuperar el error.
	58	File already exists El "nombrearchivo" especificado en una sentencia NAME es idéntico al que ya se usa en el disco.
	61	Disk full Está usándose todo el espacio de almacenamiento en disco.
	62	Input past end Se ejecuta una sentencia INPUT después de ingresados todos los datos del archivo o para un archivo nulo (vacío). Para evitar este error use la función EOF para detectar el fin del archivo.
	63	Bad record number En una sentencia PUT o GET, el número de registro es mayor que el máximo permitido (32767) o igual a cero.
	64	Bad file name Se usa un formato inválido para el "nombrearchivo" con LOAD, SAVE, KILL u OPEN (por ejemplo, un "nombrearchivo" con demasiados caracteres).
	66	Direct statement in file Se detecta una sentencia directa al cargar (LOAD) un archivo en formato ASCII. Termina la carga.
	67	Too many files Se intenta crear un nuevo archivo (usando SAVE u OPEN) estando llenas todas las 255 entradas del directorio.

ANEXO G

FUNCIONES MATEMATICAS

Funciones Derivadas

Las funciones no intrínsecas al BASIC-80 pueden calcularse de la siguiente manera:

Función	Equivalente BASIC-80
SECANTE	SEC(X)=1/COS(X)
COSECANTE	CSC(X)=1/SIN(X)
COTANGENTE	COT(X)=1/TAN(X)
SENO INVERSO	ARCSIN(X)=ATN(X/SQR(−X*X+1))
COSENO INVERSO	ARCCOS(X)=−ATN(X/SQR(−X*X+1))+1.5708
SECANTE INVERSA	ARCSEC(X)=ATN(X/SQR(X*X−1)) +SGN(SGN(X)−1)*1.5708
COSECANTE INVERSA	ARCCSC(X)=ATN(X/SQR(X*X−1)) +(SGN(X)−1)*1.5708
COTANGENTE INVERSA	ARCCOT(X)=ATN(X)+1.5708
SENO HIPERBOLICO	SINH(X)=(EXP(X)−EXP(−X))/2
COSENO HIPERBOLICO	COSH(X)=(EXP(X)+EXP(−X)/2
TANGENTE HIPERBOLICA	TANH(X)=EXP(−X)/EXP(X)+EXP(−X))*2+1
SECANTE HIPERBOLICA	SECH(X)=2/(EXP(X)+EXP(−X))
COSECANTE HIPERBOLICA	CSCH(X)=2/(EXP(X)−EXP(−X))
COTANGENTE HIPERBOLICA	COTH(X)=EXP(−X)/(EXP(X)−EXP(−X))*2+1
SENO HIPERBOLICO INVERSO	ARCSINH(X)=LOG(X+SQR(X*X+1))
COSENO HIPERBOLICO INVERSO	ARCCOSH(X)=LOG(X+SQR(X*X−1))
TANGENTE HIPERBOLICA INVERSA	ARCTANH(X)=LOG((1+X)/(1−X))/2
SECANTE HIPERBOLICA INVERSA	ARCSECH(X)=LOG((SQR(−X*X+1)+1)/X)
COSECANTE HIPERBOLICA INVERSA	ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X
COTANGENTE HIPERBOLICA INVERSA	ARCCOTH(X)=LOG((X+1)/(X−1))/2